

# **Behavior-based Control for Service Robots inspired by Human Motion Patterns**

A robotic shopping assistant

Zur Erlangung des akademischen Grades eines

**Doktors der Ingenieurwissenschaften**

von der Fakultät für Informatik  
des Karlsruher Instituts für Technologie (KIT)  
Universität des Landes Baden-Württemberg  
und nationales Forschungszentrum  
in der Helmholtz-Gesellschaft

genehmigte

**Dissertation**

von

**Michael Göller**

aus Köln-Porz

Tag der mündlichen Prüfung: 21.Oktober 2013

Erster Gutachter: Prof. Dr.-Ing. Rüdiger Dillmann

Zweiter Gutachter: Univ.-Prof. Dr.-Ing. Horst-Michael Groß



## Danksagung

Die vorliegende Arbeit ist während meiner Tätigkeit als wissenschaftlicher Mitarbeiter am *Forschungszentrum Informatik* in der Abteilung *Interaktive Diagnose und Servicesystem* entstanden. An dieser Stelle möchte ich dem damaligen Abteilungsleiter und heutigen Direktor Marius Zöllner und seinem Nachfolger Thilo Kerscher, der auch schon meine Diplomarbeit betreut hatte, dafür danken, dass sie mir diese große Chance ermöglicht haben.

Mein besonderer Dank für die Betreuung meiner Arbeit gilt meinem Doktorvater Professor Dillmann und sowie meinem Zweitgutachter Professor Groß von der *TU Ilmenau*, der mich schon als Reviewer in meinem ersten Projekt mit seinem Feedback auf den richtigen Weg brachte und dessen Arbeiten am Einkaufs-Assistenzroboter *TOOMAS* für mich immer ein großes Vorbild gewesen sind.

Danken möchte ich weiterhin einer Reihe von Personen, die wichtige Inspirationen für diese Arbeit geliefert haben: Malco Blümel, ein Informatiker mit Begeisterung für Verkehrswissenschaften, der mich auf die Simulation von Fußgängerströmen aufmerksam machte, Helge Hüttenrauch von der *Kungliga Tekniska Högskolan* (KTH, Stockholm), der mich schon früh auf die Schnittmenge meiner Gedanken mit T.B. Sheridans Veröffentlichungen hinwies, Cristian Bogdan (ebenfalls von der KTH) für die fruchtbaren Diskussionen zum Thema *human factors* in der Robotik, sowie Jan Albiez, der mich mit seiner Vorlesung an der Universität Karlsruhe für *Biologisch motivierte Robotersysteme* und *verhaltensbasierte Steuerungen* begeistert hat.

Glücklicherweise konnte meine Arbeit außergewöhnlich viele Studenten begeistern. Ohne sie wäre die Implementierung einer solch umfangreichen Steuerung niemals möglich gewesen.

Besonderer Dank gebührt zwei Studenten, die mich fast über die gesamte Zeit - als Studienarbeiter, dann als wissenschaftliche Hilfskraft und schließlich mit ihrer Diplomarbeit - unterstützt haben: Fabian Stehle und Florian Steinhardt, der schließlich auch zu einem Kollegen wurde.

Auch der Aufbau des Roboters InBOT wäre ohne die tatkräftige Unterstützung durch mehrere Praktikanten und Hiwis, ganz besonders durch Christian Billet und Anton Gorbunov, nicht möglich gewesen. Ganz besonderen Dank an dieser Stelle auch an Rolf Stober, der unzählige Komponenten für InBOT sowie für meine weiteren Projekte gefertigt hat.

Karlsruhe, Januar 2014

*Michael Göller*



## Zusammenfassung

Ziel dieser Arbeit ist es zu zeigen, dass eine verhaltensbasierte Steuerung für Serviceroboter in komplexen Anwendungen geeignet ist. Hierfür wurde, unter Verwendung menschenähnlicher Bewegungsmuster und eines verhaltensbasierten Ansatzes, eine Steuerung für mobile Serviceroboter entwickelt, welche Aufgabenplanung, globale und lokale Navigation in dynamischen Umgebungen sowie die gemeinsame Aufgabenausführung mit einem Benutzer umfasst. Eine verhaltensbasierte Steuerung im Allgemeinen, und das hier verwendete Verhaltensnetzwerk im Speziellen, besteht – im Gegensatz zu einem klassischen allumfassenden Planer – aus einer Vielzahl von unabhängigen Modulen, die alle eine individuelle Aufgabe verfolgen. Das Gesamtverhalten des Systems ergibt sich aus der Vereinigung der Einzelverhalten („Emergenz“).

Als roter Faden für die Entwicklung und Evaluation soll das Supermarkt-Szenario dienen. Daher wurde, parallel zur Entwicklung der Steuerung, der Einkaufswagen-Roboter *InBOT* „*Interactive Behavior-Operated shopping Trolley*“ aufgebaut. Im Zuge des Entwicklungsprozesses mussten drei Teilziele erarbeitet werden:

1. Die verhaltensbasierte Steuerung: Es wurde eine verhaltensbasierte Steuerung entworfen, die den Anforderungen an einen Service Roboter in einer dynamischen Umgebung genügt, im Besonderen hinsichtlich der Navigation und der Benutzerinteraktion. Die Steuerung verfügt über die nötige Robustheit und Berechenbarkeit, um Aufgaben erfüllen zu können, ohne dabei den Vorteil der Flexibilität und Modularität einzubüßen. Es konnte gezeigt werden, dass sich gerade ein verhaltensbasierter Ansatz sehr gut eignet, um die Herausforderungen des Szenarios zu meistern.

2. Das Navigationskonzept: Es wurde ein Navigationskonzept entworfen, das den Roboter in einem dynamischen Umfeld sicher navigieren lässt. Neben der lokalen und globalen Pfadplanung kann mit mobilen Hindernissen sicher umgegangen werden. Das Navigationssystem kommt ohne globale metrische Karte aus, und ist somit von Änderungen in der Umgebung unabhängig.

3. Die Anwendung: Die Steuerung ermöglicht es dem Roboter Aufgaben für – und mit – dem Menschen zu erledigen. Steuerung und Benutzer üben hierbei die tatsächliche Kontrolle über die Bewegung des Roboters gemeinsam aus („*control sharing*“), anstatt – wie häufig vorgefunden – in einem Zyklus aus abwechselndem Befehl und Ausführung zu arbeiten („*control trading*“). Um dem Benutzer die Fähigkeiten des Roboters möglichst flexibel zur Verfügung zu stellen, wurden fünf Steuerungsmodi realisiert: *autonomous*, *guiding*, *following*, *servoing* und *manual control*. Die Anforderungen der Anwender wurde im Entwicklungsprozess berücksichtigt: Einige der high-level Requirements resultieren aus einer Umfrage; außerdem wurde das fertige System in Anwenderstudien evaluiert.

Die zentrale Inspiration für die Steuerung kommt von einem Modell des Bewegungsverhaltens von Fußgängern, das in Form einer hybriden, hierarchischen Steuerungsarchitektur umgesetzt wurde. Die einzelnen funktionalen Module sind in drei Schichten zusammengefasst: in die strategische, die taktische und die reaktive Schicht. Ergänzt werden diese drei funktionalen Schichten durch eine anwendungsspezifische Kommunikationsschicht und eine plattformabhängige Hardware-Abstraktionsschicht. Da die globale Navigation der strategischen Schicht auf einer topologischen Navigation aufbaut, kommt die Steuerung ohne eine globale metrische Karte aus, die ständig aktualisiert werden müsste.



## Kurzfassung

Bisher werden Serviceroboter meist getrennt von ungeschulten Menschen eingesetzt und dürfen nur von speziell ausgebildetem Personal bedient werden. Der Einsatz erfolgt in einer möglichst statischen Umgebung, um dem Roboter die Navigation zu erleichtern und eine sonst notwendige, regelmäßige Anpassung der gespeicherten Karte zu vermeiden.

Beispiele können etwa Flurförderfahrzeuge sein, die in Krankenhäusern in den Versorgungsgängen eingesetzt werden und dort Lebensmittel und Wäsche liefern. In den Versorgungsgängen dürfen sich nur Angestellte aufhalten, jedoch keine Patienten. Auch bei Reinigungs- oder Inspektionsrobotern wird der Mensch eher als Hindernis betrachtet statt als Partner.

Ziel dieser Dissertation ist es zu zeigen, dass eine verhaltensbasierte Steuerung geeignet ist, um Serviceroboter in komplexe Anwendungen in öffentlichen Bereichen einzubringen. Hierfür wurde unter Verwendung eines verhaltensbasierten Ansatzes, eine Steuerung für mobile Serviceroboter entwickelt, die Aufgabenplanung, globale und lokale Navigation in dynamischen Umgebungen sowie die gemeinsame Aufgabenausführung zusammen mit einem Benutzer umfasst. Die zentrale Inspiration kommt hierbei von einem Modell des Bewegungsverhaltens von Fußgängern, das in Form einer hybriden, hierarchischen Steuerungsarchitektur umgesetzt wurde.

Aufgrund seiner Komplexität und Dynamik diente ein spezielles Szenario bei der Entwicklung der Steuerung als roter Faden: Einkaufen in einem Supermarkt, mit Unterstützung durch den im Rahmen dieser Arbeit entwickelten Roboter *InBOT* („*Interactive Behavior-Operated shopping Trolley*“). Hierfür wurden, ergänzend zur Kernsteuerung, exemplarisch anwendungsspezifische Fähigkeiten entwickelt, oder bestehende integriert, um die Steuerung im Kontext dieses Anwendungsszenarios testen zu können. Um bei der Evaluation die nötige Tiefe zu erreichen, wurde für dieses Szenario ein im Labor nachgebildeter, kleiner Supermarkt genutzt, in dem *InBOT* von nicht trainierten Benutzern als Einkaufshilfe verwendet wurde.

Das gewählte Szenario stellt viele Herausforderungen an den Roboter und an die Steuerung: Der Roboter soll groß genug sein, um Waren transportieren zu können, dabei jedoch trotz seiner Größe und Form in der Lage sein, in engen Gängen voller Hindernisse operieren zu können. Hierfür muss der Roboter, gleich einem normalen Einkaufswagen, über ein holonomes Fahrwerk verfügen. Das bedeutet jedoch, dass der Gefahrenbereich des Roboters nicht nur vorne und hinten ist, sondern volle 360 Grad umfasst. Hinzu kommt, dass Supermärkte viele nicht-statische, sich zum Teil sogar selbst bewegende, Hindernisse beherbergen, die sowohl die Navigation des Roboters wie auch die Kartierung der Umgebung behindern. Und schließlich muss der Roboter, entsprechend seiner Rolle als Einkaufsassistent, alle Aktionen in Kooperation mit dem Benutzer durchführen.

Um das Ziel dieser Arbeit zu erreichen wurden Lösungen in drei Themenfelder erarbeitet:

- 1. Die verhaltensbasierte Steuerung:** Es wurde eine verhaltensbasierte Steuerung entworfen, die den Anforderungen an einen Service Roboter in einer dynamischen Umgebung genügt, insbesondere hin-

sichtlich der Navigation und der Benutzerinteraktion. Die Steuerung weist die nötige Robustheit und Berechenbarkeit auf, um Aufgaben erfüllen zu können, ohne dabei den Vorteil der Flexibilität und Modularität einzubüßen. Es wurde gezeigt, dass sich gerade ein verhaltensbasierter Ansatz sehr gut eignet um die Herausforderungen des Szenarios zu meistern.

- 2. Das Navigationskonzept:** Es wurde ein Navigationskonzept entworfen, das den Roboter in einem dynamischen Umfeld sicher navigieren lässt. Neben der lokalen und globalen Pfadplanung kann mit mobilen Hindernissen sicher umgegangen werden. Außerdem kommt das Navigationssystem ohne globale metrische Karte aus, und ist somit von Änderungen in der Umgebung unabhängig. Das Navigationssystem dieser Arbeit lehnt sich stark an das Bewegungs- und Navigationsverhalten von Fußgängern an, da diese bekannter Weise in der Lage sind, alle Herausforderungen des Szenarios zu meistern.
- 3. Die Anwendung:** Die Steuerung ermöglicht es dem Roboter Aufgaben für – und mit – dem Menschen zu erledigen. Daher waren Roboterverhalten im Gebiet der Mensch-Roboter-Interaktion ein Schwerpunkt bei der Entwicklung der Steuerung. Dadurch sind Steuerung und Benutzer in der Lage, die tatsächliche Kontrolle über die Bewegung des Roboters gemeinsam auszuüben („*control sharing*“). Diese Art der Kontrolle unterscheidet sich grundsätzlich von der gängigen Vorgehensweise, bei der der Mensch den Befehl vorgibt und der Roboter ihn danach autonom ausführt („*control trading*“).

Damit der Benutzer die Fähigkeiten des Roboters in verschiedenen Situationen optimal nutzen kann, werden ihm über ein multimodales Benutzer-Interface fünf Kontroll-Modi zur Verfügung gestellt:

- **Autonomous Mode:** Der Roboter erledigt die Aufgabe unabhängig vom Benutzer.
- **Guiding Mode:** Der Roboter führt den Benutzer an ein Ziel und passt sich dabei an die Bewegung des Benutzers an, das heißt, er hält immer Kontakt und wartet auf den Benutzer.
- **Following Mode:** Der Roboter folgt dem Benutzer, navigiert aber selbstständig um Hindernisse herum.
- **Servoing Mode:** Der Roboter „imitiert“ die Bewegung des Benutzers.
- **Manual Steering Mode:** Der Benutzer steuert den Roboter direkt durch ein kraftsensitives Eingabegerät. Auch hierbei navigiert der Roboter selbstständig um Hindernisse herum.

Das Konzept für die Steuerung ist durch verschiedene Arbeiten inspiriert. Zunächst ist das Modell von Fußgängerbewegungen zu nennen, das von S.P. Hoogendoorn im Kontext der Simulation von Evakuierungsszenarien aufgestellt wurde. Dieses Modell wurde mit klassischen Konzepten für hierarchische Steuerungsarchitekturen fusioniert. Die unteren beiden Ebenen der Architektur wurden durch eine verhaltensbasierte Steuerung realisiert, die von den Verhaltensnetzwerken, die J. Albiez zum Steuern von Laufmaschinen entworfen hat, inspiriert ist. Orthogonal zur klassischen Top-Down-Konstruktion der in Ebenen organisierten Architekturen wurde das Konzept des *control sharing* in der Architektur umgesetzt.

Die resultierende dreischichtige Architektur vereint reaktive und deliberative Verhalten in den unteren sowie einen Aufgabenplaner mit integrierter topologischer Navigation in den oberen Schichten. Die unterschiedlichen Ebenen ermöglichen es, unterschiedliche Anforderungen an die Antwortzeiten zu berücksichtigen und einfache von komplexen Aufgaben zu trennen. Zwischen den Schichten befinden sich Schnittstellen, um die Kapselung und somit die Modularisierung zu ermöglichen. In den unteren Schichten sorgen im



Framework MCA2 implementierte Verhalten der verhaltensbasierten Steuerung für die geforderte Reaktivität des Systems sowie für ein hohes Maß an Modularität, Rekombinierbarkeit und Erweiterbarkeit.

Die drei von Hoogendoorn inspirierten Schichten sind ergänzt um eine Kommunikationsschicht oberhalb und eine Hardware-Abstraktionsschicht unterhalb der drei funktionalen Schichten. Passend zu den Abstraktionsebenen der Befehlsverarbeitung stellt das ebenfalls hierarchische Umweltmodell Daten in verschiedenen Abstraktionsgraden zur Verfügung. Außerdem dient das Weltmodell als Schnittstelle zu anderen Robotersystemen sowie zu in der Umgebung eingebetteten Sensoren.

Die fünf Schichten sind im Folgenden kurz zusammengefasst:

**Kommunikationsschicht:** Dies ist die anwendungsspezifische Schicht. Sie dient der Befehligung des Roboters durch einen Benutzer. Im Falle des Einkaufsroboters *InBOT* findet sich hier ein Touchscreen-basiertes graphisches Benutzerinterface mit Sprachausgabe, sowie ein Barcode-Reader. Eingaben dieser Modalitäten werden mittels einer Applikationslogik und einer Produktdatenbank in Aufgaben wie Fahrbefehle oder Moduswechsel umgewandelt. Über ein definiertes Interface gibt die Kommunikationsschicht diese Aufgaben an die strategische Schicht weiter.

**Strategische Schicht:** Die strategische Schicht akzeptiert Aufgaben (z.B.: Fahraufträge, Moduswechsel, etc.) von der Kommunikationsschicht über ein definiertes Protokoll auf TCP-Basis. Sie arbeitet mit Hilfe einer flexiblen Baumstruktur für die Aufgabenverwaltung sowie einer topologischen Karte für die globale Navigation. Letztere enthält metrische Annotationen in Form der relativen Koordinaten der Übergänge zwischen benachbarten topologischen Gebieten. Aus einer durch das Interface von der Kommunikationsschicht empfangenen Aufgabe – meist ein Fahrauftrag – wird das gewünschte Ziel extrahiert. Die topologische Navigation plant sodann eine Sequenz von topologischen Gebieten, die passiert werden müssen, um das Ziel zu erreichen. Daraufhin verfeinert die Routenplanung diesen Plan, indem sie lokale Koordinaten berechnet, die der Roboter passieren muss, um sein aktuelles topologisches Gebiet an der richtigen Stelle zu verlassen. Hierbei betritt der Roboter automatisch das benachbarte topologische Gebiet, woraufhin der Verfeinerungsschritt wiederholt wird. Diese topologischen Navigationspunkte werden in der Baumstruktur zusammen mit den Aufgaben gespeichert und der jeweils aktuelle Punkt an die nächst tiefere Schicht weitergegeben.

**Taktische Schicht:** Die taktische Schicht akzeptiert lokale Koordinaten eines anzufahrenden Ziels von der strategischen Schicht. Diese Koordinaten werden im Zuge einer geometrischen Szenen-Analyse durch höhere Verhalten mit der Robotergeometrie und einer Liste von segmentierten Objekten, inklusive deren Eigenschaften, verglichen. Basierend auf statischen Objekten (d.h. Hindernissen) werden Sub-Ziele definiert, die den Roboter auf einem effizienten Pfad um diese herum steuern. Im Falle von sich bewegenden Objekten wird deren wahrscheinlicher Pfad prediziert und es werden wiederum Sub-Ziele berechnet, die den Roboter aus der Gefahrenzone steuern. Die Koordinaten des resultierenden Sub-Ziels werden an die reaktive Schicht weitergereicht.

**Reaktive Schicht:** Die reaktive Schicht akzeptiert lokale Zielkoordinaten von der taktischen Schicht. Sie erfüllt drei Hauptaufgaben, die durch reaktive Verhalten implementiert sind: Der Roboter bewegt sich in Richtung des (Sub-)Ziels, er weicht Hindernissen wie auch sich bewegenden Objekten aus und im Notfall stoppt er, um Kollisionen zu verhindern. Hierfür kann die reaktive Schicht auf die Objektliste mit der Bewegungsprediktion, auf eine lokale Belegtheitskarte, sowie auf die Rohdaten von Entfernungssensoren für eine besonders schnelle Reaktion zurückgreifen. All diese Funktionalitäten werden

von einzelnen Verhalten oder Verhaltensgruppen realisiert, die eine gemeinsame Sprache sprechen: Sie erzeugen je einen 3D-Sollgeschwindigkeits-Vektor, bestehend aus Translation in X und Y Richtung, sowie der Orientierung. Diese Vektoren werden in mehreren Schritten fusioniert. Der resultierende Vektor wird an die Hardwareabstraktionsschicht weitergereicht, die ihn dann auf der Plattform ausführt.

**Hardware Abstraktionsschicht:** Dies ist die plattformabhängige Schicht. Im Falle des Roboters *InBOT* kapselt sie einen holonomen Mecanum Antrieb, aber zum Beispiel im Falle des Roboters *Odete* einen Differentialantrieb, so dass 3D Sollgeschwindigkeits-Vektoren  $(X, Y, \Theta)$  ausgeführt werden können. Außerdem kapselt sie die Sensoren für Odometrie und globale Positionskorrekturen sowie die jeweiligen Sensoren zur Hindernisdetektion. Letztere werden dann in einer Belegtheitskarte zusammengeführt.

Die vorliegende Arbeit ist wie folgt aufgebaut:

Das erste Kapitel beschreibt Motivation und Ziel der Arbeit und erarbeitet high-level Requirements. Kapitel zwei untersucht den Stand der Technik im Bereich der Steuerungsarchitekturen, Navigationsmethoden und der Serviceroboter.

Basierend auf den Ergebnissen der ersten beiden Kapitel wird in Kapitel drei der Entwurf der hierarchischen Steuerungsarchitektur beschrieben, die auf dem Modell des menschlichen Bewegungsverhaltens, dem *control sharing* sowie den Verhaltensnetzwerken beruht.

Kapitel vier beschreibt wie in den einzelnen Schichten der Steuerungsarchitektur das Navigationskonzept umgesetzt wurde, angefangen bei den Sicherheitsverhalten über reaktive Verhalten und die geometrische Szenenanalyse bis hin zur Aufgabenplanung.

Die Kapitel fünf, sechs und sieben beschreiben jeweils Konzepte, die mehrere Schichten überspannen: die Handhabung sich bewegender Hindernisse, die Benutzerinteraktion und die Multi-Roboter-Koordination, die auch mit Hilfe der verhaltensbasierten Architektur realisiert wurden.

Kapitel acht fasst die Arbeit zusammen und diskutiert die Ergebnisse und offene Punkte.

In den Anhängen werden ergänzenden Komponenten beschrieben, die den Rahmen der Arbeit sprengen würden, wie zum Beispiel das Benutzerinterface, kamerabasiertes Objekttracking oder die Entwicklung des Roboters *InBOT* selbst. Außerdem werden Robotersysteme beleuchtet, in denen Komponenten dieser Arbeit integriert wurden oder werden.

Zusammenfassend kann festgehalten werden, dass erstmals eine verhaltensbasierte Steuerung entwickelt wurde, die es einem Roboter ermöglicht, Aufgaben zusammen mit einem Benutzer in einer dynamischen, von Menschen frequentierten Umgebung zu erfüllen. Hierzu wurden bei Menschen beobachtete Verhaltensmuster mit Hilfe von sogenannten Verhaltensnetzwerken im Framework MCA2 realisiert. Die Leistungsfähigkeit der Steuerung wurde in Anwenderstudien unter Beweis gestellt.

Da die globale Navigation auf einer topologischen Navigation aufbaut, wird eine maßgebliche Herausforderung dynamischer Umgebungen gelöst: die Steuerung kommt ohne globale metrische Karte aus, die ständig aktualisiert werden müsste. Stattdessen übernimmt die topologische Navigation den globalen Anteil der Navigation, während die verhaltensbasierte Steuerung, die nur auf dem aktuell einsehbaren Bereich arbeitet, für den lokalen Teil zuständig ist.

## Abstract

The goal of this thesis is to demonstrate that a service robot's control system based on human motion patterns implemented by *Behavior Networks* – a special case of a *Behavior-Based Control* – performs well in complex scenarios involving dynamic environments and human robot interaction, while providing significant benefits in orchestrating present abilities and integrating new functionalities into the system.

In contrast to conventional approaches based on monolithic trajectory planners, a *Behavior-Based Control* consists of a large number of independent behavior modules with individual tasks – the system's overall behavior *emerges* from the individual behaviors.

Providing shopping assistance in supermarkets is the predominant theme of this thesis. Thus, the robotic shopping cart *InBOT* (“*Interactive Behavior-Operated shopping Trolley*”) has been designed to apply the developed control system.

During the development of the behavior-based control system, three major topics were addressed:

Firstly the *behavior-based* control system: A control architecture was designed based on *Behavior Networks* which fulfills the demands posed on a *service robot* in a dynamic environment, in particular navigation and interaction with a human user. While assuring the required robustness and predictability, the control system has to preserve its modularity, flexibility, and extensibility.

Secondly the navigation system: As well as getting to the given target, the navigation system has to be able to cope with a highly dynamic environment including moving obstacles. Additionally, it shall not depend on a global metrical map in order to be independent from changes in the environment, which can be expected to take place frequently.

And thirdly the application of the system: As the control system has to facilitate the cooperative task execution of the robot and the user, both have to be able to simultaneously exert control on the robot's action (“*control sharing*”). This is in contrast to the traditional approach of giving orders to the robot and then waiting while the robot executes the task autonomously (“*control trading*”). To facilitate the flexible utilization of the robot's abilities, five *modes of operation* have been implemented: *autonomous*, *guiding*, *following*, *servoing*, and *manual steering*. The potential users were taken into consideration during the development process: several high-level requirements have been derived from a survey and the complete system has been evaluated in user studies at the end of this work.

Probably the most important source of inspiration for the design of the presented hierarchical and hybrid control architecture is a model of pedestrian motion behaviors. The three functional layers (strategic, tactical and reactive) of the architecture contain the various behavior modules. These three generalized layers are augmented by an application-specific communication layer and a platform-specific hardware abstraction layer. The global navigation of the strategic layer is based on topological maps and thus is independent of local changes to the surroundings, local navigation being assured by the *Behavior Network* which works based on current sensor readings, only.



# Contents

Danksagung . . . . .	iii
Zusammenfassung . . . . .	v
Kurzfassung . . . . .	vii
Abstract . . . . .	xi
<b>1 Introduction . . . . .</b>	<b>1</b>
1.1 Motivation . . . . .	1
1.2 Goal of this thesis . . . . .	3
1.3 Recurring theme: scenario supermarket . . . . .	5
1.3.1 Infrastructure of the supermarket . . . . .	5
1.3.2 Challenges posed by the supermarket scenario . . . . .	6
1.3.3 Survey with customers at the supermarkets . . . . .	8
1.4 High-level requirements for the control system . . . . .	12
1.4.1 R2 Human-Robot Interaction capability: user interaction . . . . .	13
1.4.2 R8 Generalization . . . . .	15
1.5 Approach for the <i>service robot</i> 's control system . . . . .	16
1.5.1 Concept of the control architecture . . . . .	16
1.5.2 Application-specific part of the control system . . . . .	17
1.5.3 Evaluation of the control system . . . . .	17
1.6 Contribution . . . . .	18
1.7 Organization of this work . . . . .	18
<b>2 State of the Art . . . . .</b>	<b>21</b>
2.1 Brief introduction to pedestrian motion modeling and potential applications in robotics	21
2.2 Control architectures for mobile robots . . . . .	26
2.2.1 Deliberative or functional architectures . . . . .	26
2.2.2 Reactive architectures . . . . .	26
2.2.3 Behavior-based architectures . . . . .	27
2.2.4 Hybrid and hierarchical architectures . . . . .	34
2.3 Navigation, self-localisation and mapping . . . . .	37
2.3.1 RFID-based self-localization . . . . .	38
2.3.2 Maps and mapping . . . . .	39
2.3.3 Collision avoidance . . . . .	40
2.3.4 Path planning . . . . .	44

2.4	Service robots . . . . .	44
2.4.1	Functional service robots . . . . .	44
2.4.2	Social service robots . . . . .	46
2.4.3	Shopping assistants and shopping robots . . . . .	49
2.5	Discussion and resume . . . . .	54
<b>3</b>	<b>The Hybrid Control Architecture . . . . .</b>	<b>57</b>
3.1	Inspiration for the control architecture . . . . .	57
3.1.1	Model of pedestrian's motion patterns as blueprint of the navigation system . . . . .	58
3.1.2	Hybrid and layered architecture as organizational paradigm . . . . .	58
3.1.3	<i>Behavior Networks</i> for the implementation of reactive behaviors . . . . .	59
3.1.4	The concept of <i>control sharing</i> and <i>control trading</i> to enhance user interaction . . . . .	60
3.2	Design of the control architecture . . . . .	60
3.2.1	Design Criteria of the control architecture . . . . .	61
3.2.2	Fundamental concepts of the control architecture . . . . .	61
3.2.3	Incorporation of the components . . . . .	63
3.3	The resulting control architecture . . . . .	65
3.3.1	The architecture of the infrastructure . . . . .	66
3.3.2	The <i>local world model</i> . . . . .	66
3.3.3	Interfaces . . . . .	68
3.3.4	Data flow . . . . .	70
3.3.5	Integration with the MCA2 framework using <i>UComs</i> . . . . .	70
3.4	Exemplary system architectures . . . . .	70
3.5	Summary of the control architecture . . . . .	71
<b>4</b>	<b>BBC: Navigation, Obstacle Avoidance and Safety . . . . .</b>	<b>73</b>
4.1	Navigation system and control architecture . . . . .	73
4.1.1	The <i>communication layer</i> . . . . .	75
4.1.2	The <i>strategic layer</i> . . . . .	76
4.1.3	The <i>tactical layer</i> . . . . .	76
4.1.4	The <i>reactive layer</i> . . . . .	77
4.1.5	The <i>Hardware Abstraction Layer</i> . . . . .	79
4.2	Summary of the self-localization concept . . . . .	80
4.2.1	Odometry for local self-localization . . . . .	80
4.2.2	Global self-localization by floor-mounted RFID barriers . . . . .	81
4.3	The <i>behavior-based</i> control system implementing the tactical and reactive layers . . . . .	81
4.3.1	Architecture of the <i>Behavior-Based Control</i> . . . . .	82
4.4	The reactive behaviors (RB) . . . . .	84
4.4.1	RB: Safety behavior . . . . .	85
4.4.2	RB: Behaviors for adapting the velocity to the task and the user . . . . .	87
4.4.3	RB: <i>Avoid Obstacles (AO)</i> – behaviors for the reactive obstacle avoidance of static obstacles . . . . .	88
4.4.4	RB: Look for gaps . . . . .	98

4.4.5	RB: Follow Wall . . . . .	99
4.4.6	RB: Break Tie . . . . .	99
4.4.7	RB: Orientation of the robot . . . . .	100
4.4.8	RB: Tasks-oriented input behaviors . . . . .	100
4.5	The <i>tactical behaviors</i> . . . . .	102
4.5.1	TB: Look for Corners – geometry-based obstacle avoidance and local navigation . . . . .	102
4.5.2	TB: Follow Moving Object . . . . .	105
4.5.3	TB: Virtual Train . . . . .	105
4.5.4	TB: Force Commands . . . . .	106
4.5.5	TBG: Geometrical Scene Analysis – <i>tactical behaviors</i> focusing on adapting to the dynamic environment . . . . .	106
4.6	The <i>strategic layer</i> – global navigation and planning . . . . .	113
4.6.1	Topologic-metric map . . . . .	114
4.6.2	Global planning . . . . .	114
4.6.3	Evaluation of global planning . . . . .	116
4.7	Application logic and <i>communication layer</i> . . . . .	116
4.8	Experiments and evaluation . . . . .	117
4.8.1	First system test . . . . .	117
4.8.2	Second system test . . . . .	117
4.8.3	Third system test . . . . .	118
4.9	Discussion . . . . .	120
<b>5</b>	<b>Avoiding Collisions with Moving Objects . . . . .</b>	<b>123</b>
5.1	Reactive avoidance of moving objects . . . . .	125
5.1.1	RB: The Escape Behavior . . . . .	125
5.1.2	RB: The Evade Behavior . . . . .	126
5.1.3	Experimental results . . . . .	127
5.2	Proactive avoidance of moving objects using spatio-temporal plans . . . . .	130
5.2.1	Baseline: Data and pre-processing . . . . .	130
5.2.2	Spatio-temporal calculation of safe path using an $A^*$ algorithm . . . . .	131
5.2.3	Optimization of path . . . . .	132
5.2.4	Utilization of the <i>Behavior-Based Control</i> . . . . .	133
5.2.5	Experimental results using the planner . . . . .	133
5.3	Experiments and tests . . . . .	134
5.3.1	Comparison of components . . . . .	134
5.3.2	Stress test . . . . .	134
5.3.3	Overall success rate . . . . .	136
5.3.4	System in application . . . . .	137
5.4	Discussion . . . . .	138
<b>6</b>	<b>User Interaction . . . . .</b>	<b>141</b>
6.1	Five <i>modes of operation</i> . . . . .	143

6.2	Modalities for interaction and the multimodal user interface . . . . .	145
6.3	Mode transitions . . . . .	146
6.4	Interaction regarding modalities . . . . .	147
6.4.1	Interaction based on force input by the user . . . . .	147
6.4.2	Interaction based on the observation of the user . . . . .	150
6.4.3	Interaction based on high level user interface . . . . .	152
6.5	Tactical behaviors for adapting to and communicating with the user . . . . .	153
6.6	Sharing and trading of control . . . . .	158
6.6.1	Force-based <i>control sharing</i> . . . . .	160
6.6.2	Observation-based <i>control sharing</i> . . . . .	160
6.6.3	Command-based <i>control trading</i> . . . . .	161
6.6.4	Impact of <i>control sharing</i> while operating in the individual modes . . . . .	162
6.7	Experiments . . . . .	162
6.7.1	Experimental evaluation of <i>control sharing</i> and <i>control trading</i> . . . . .	163
6.7.2	Evaluation of <i>InBOT</i> 's behavior by potential users . . . . .	166
<b>7</b>	<b>Multi-Robot Coordination . . . . .</b>	<b>171</b>
7.1	Sharing of the <i>local world model</i> . . . . .	172
7.2	Multi-robot path arranging behaviors (RB, TB) . . . . .	172
7.3	TB: Queueing up . . . . .	173
7.4	TB: Virtual train . . . . .	174
<b>8</b>	<b>Conclusion, Discussion, and Open Issues . . . . .</b>	<b>177</b>
8.1	Summary . . . . .	177
8.2	Implementation on multiple robots . . . . .	178
8.2.1	Implementation on <i>InBOT</i> . . . . .	179
8.2.2	Implementation on <i>ETrolley</i> . . . . .	179
8.2.3	Implementation on <i>Odete</i> . . . . .	180
8.2.4	Implementation on <i>LAURON</i> . . . . .	181
8.2.5	Implementation on <i>CityPod</i> . . . . .	181
8.2.6	Implementation on <i>HoLLiE</i> . . . . .	182
8.3	Discussion . . . . .	182
8.3.1	Behavior-based control system . . . . .	183
8.3.2	Dynamic environments . . . . .	184
8.3.3	User interaction and <i>control sharing</i> . . . . .	185
8.4	Open issues . . . . .	186
8.5	Roll out: from the lab to application . . . . .	187
<b>A</b>	<b>Interdependent Work, List of Publications, and Student's Theses . . . . .</b>	<b>189</b>
	List of Publications . . . . .	191
	List of Students' Theses . . . . .	193



<b>B</b>	<b>The Interactive Behavior-Operated Trolley (InBOT), InBOT-2, and ETrolley . . . . .</b>	<b>195</b>
B.1	The robot <i>InBOT2</i> . . . . .	196
B.2	ETrolley . . . . .	198
B.3	Electronical layout . . . . .	200
B.4	Self-localization . . . . .	202
B.4.1	Odometry for local self-localization . . . . .	202
B.4.2	Global self-localization by landmarks . . . . .	203
B.5	2D Odometry wheels . . . . .	205
B.6	Force sensitive handlebar . . . . .	207
B.6.1	Sensor concept . . . . .	207
B.6.2	Three iterations of the <i>force sensitive handle</i> . . . . .	209
<b>C</b>	<b>Complementary System Components . . . . .</b>	<b>213</b>
C.1	Object tracking based on the occupancy grid and planar laser scanners . . . . .	213
C.1.1	Pre-processing of the grid map . . . . .	214
C.1.2	Tracking objects . . . . .	214
C.1.3	Updating the tracking history . . . . .	214
C.2	Detecting and tracking the user using onboard sensors . . . . .	215
C.3	The <i>intelligent environment</i> . . . . .	217
C.4	The <i>InBOT-UI</i> . . . . .	221
C.5	The <i>CR-UI</i> . . . . .	224
C.6	The <i>TaPAC Client</i> . . . . .	225
C.7	The IDS graph-based navigation system . . . . .	225
	<b>List of Figures . . . . .</b>	<b>226</b>
	<b>Bibliography . . . . .</b>	<b>231</b>



# 1. Introduction

## 1.1. Motivation

In the early 20th century science-fiction authors came up with the idea of robots – intelligent machines designed to help the people. The first occurrence was in 1921 in Capek’s drama *Rossum’s Universal Robots R.U.R.* where humanoid robots are used as a cheap labor force without rights – until they start a rebellion. Later on, in the year 1927, the movie *Metropolis* (Fig. 1.1) showed, like its predecessor *R.U.R.*, robots resembling humans in appearance and abilities. In contrast, the first real robot *Unimate* (Fig. 1.2) which was developed in 1954 was only able to perform simple tasks. It served as a programmable manipulator for cutting and welding die castings in the assembly lines of *General Motors*. Since then industrial robots have made major improvements in speed, control and precision and thus form the core of many factories, especially in Japan, Germany and the USA. For example in Europe in 2007 alone 35000 new robots were installed [173]. But still, compared to the science-fiction robots from almost 100 years ago, these modern robots cannot compete in flexibility, intelligence and autonomy.

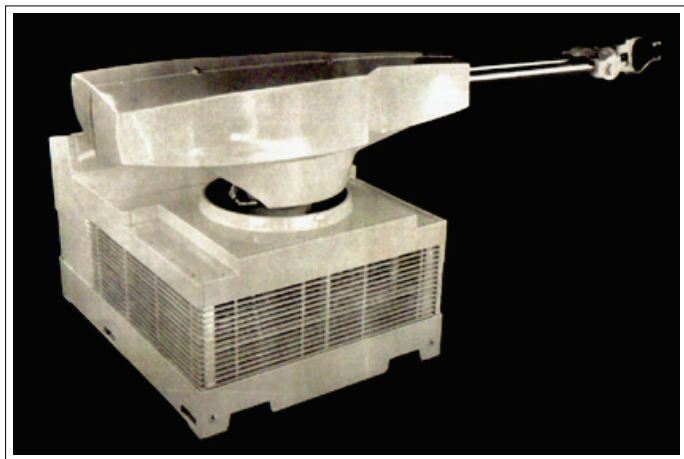


Fig. 1.1.: The movie *Metropolis* by Fritz Lang (1927) (image source: [132])

Fig. 1.2.: Industrial robot *Unimate* developed by the company Unimation (1961) (image source: [178])

Accompanying technological developments, the idea driving the science-fiction authors was taken up again: the development of robots which should serve the general populace instead of only being used in factories – the birth of the service robots. In 1994 the Fraunhofer institute defined:

“Ein Serviceroboter ist eine frei programmierbare Bewegungseinrichtung, die teil- oder vollautomatisch Dienstleistungen verrichtet. Dienstleistungen sind dabei Tätigkeiten, die nicht der direkten industriellen Erzeugung von Sachgütern, sondern der Verrichtung von Leistungen für Menschen und Einrichtungen dienen.”  
(Fraunhofer-Institut, 1994)

## 1. Introduction

---

This means that *service robots* shall perform services for the people. These services shall not – in contrast to robots in general – be aimed at producing goods.

But, up to today only specially trained people are allowed and able to operate the *service robots*. Moreover, *service robots* have to operate in special areas which are restricted to trained personnel. The environment is kept as static as possible as these robots lack a sufficiently high level of flexibility and intelligence. The static and predictable environment eases the robot's navigation and enables the robot to use a static map of the surrounding. Otherwise this map would be outdated very quickly and thus would have to be updated continuously, which is an extremely challenging task in its own – and subject of recent research.

An example of current commercial *service robots* are automated guided vehicles (AGV) like the *Telelift* robot shown in Fig. 1.3. These operate in the service corridors of hospitals and deliver goods like clothes and food. Only employees are allowed to enter these corridors and it is strictly forbidden to put anything on the ground which could be an obstacle for the robots. Another example of a commercial *service robot* is the automatic subway train deployed by *Siemens* in the city of Nürnberg (see Fig. 1.4). Here again the robot operates in a static environment and even though it transports people there is no close interaction between the people and the robot. The people don't have any influence on the task execution or the robot's behavior. Another group of *service robots* are cleaning and inspection robots. Some of these operate in public areas, even at residences like the well known vacuuming-robot *ROMBA*. If these robots meet people during their task, these are usually perceived as obstacles rather than partners for interaction. Others operate in areas where people cannot reach them anyway, like the window sweeping robot *RACOON* (Fig. 1.5). Even most entertainment robots, whose central idea is entertaining people, usually offer no real interaction. They possess a tailor made set of commands including all kinds of tricks. Once they received a command they execute it autonomously.

All this allows the following conclusion: currently there are several commercial *service robots* which are designed to perform tasks for people. But they are only able to operate properly when there is no interference from people. Hence, the robots are used only in professional environments and in collaboration with trained operators. In contrast, if the robots are to be able to carry out real services for the general public, it is necessary for *service robots* to be able to operate in public places and to interact with people. Here flexibility and adaptability are the most important keywords.



Fig. 1.3.: *TransCar* automatic guided vehicle (AGV) by Swisslog, operating in hospitals (image source: [157])



Fig. 1.4.: Automated subway train developed by Siemens, operating in Nürnberg (the RUBIN project) (image source: [150])



Fig. 1.5.: Window cleaning robot RACOON (image source: [50])

Several groups of researchers have been – and currently are – working on more flexible and adaptive

systems in order to extend the abilities of commercial *service robots*. In recent years, several field experiments were performed where these originally laboratory systems have been tried in public environments. The motivations for these complex and costly experiments range from demonstrating the developed abilities to performing studies in human-robot interaction with large quantities of untrained users. In most cases, the experimental *service robots* are used as *guiding robots* e.g. as tour guides in museums or as mobile information kiosks in hardware stores. Early examples are the robots *Rhino* [30] and some years later *MIN-ERVA* (Fig. 1.6) [163], both developed by S. Thrun in the years 1995 and 1999 respectively. Both robots have served as tourguide robots in museums for several weeks. In 2002, R. Siegwart deployed a whole fleet of robots of the type *Robox* ([163], Fig. 1.7) as tour guide robots during the *Expo02*. Later on, in 2008 and 2009, the robot *TOOMAS* (see Figure 1.8, [66]) assisted customers in a *TOOM* hardware store. It was developed by H.M. Gross from the University of Ilmenau in collaboration with the company *MetraLabs*. *TOOMAS* was based on the robot platform *SCITOS* which is also the basis for two successors of *TOOMAS*. These two robots called *ShopBot* [67] are currently operating in the *real.-future store* where they present a variety of innovations to the shop's customers. More information on these robots can be found in the chapter *state of the art* (Chapter 2).

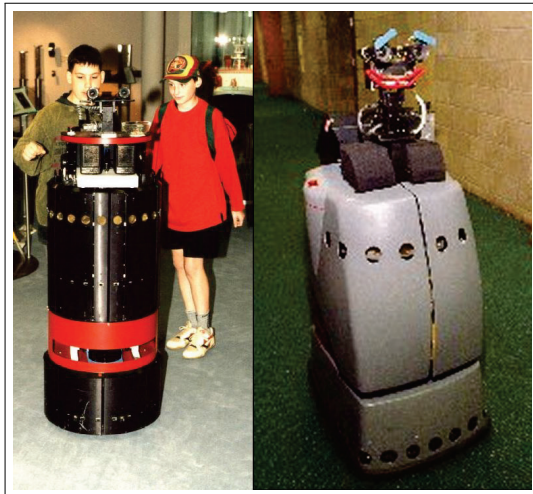


Fig. 1.6.: Rhino (image source: [172]) and MIN-ERVA (image source: [135])



Fig. 1.7.: Robox (image source: [42])



Fig. 1.8.: TOOMAS (image source: [167])

## 1.2. Goal of this thesis

The goal of this thesis can be summarized as follows: It will be demonstrated that a service robot's control system based on the biologically motivated *Behavior Networks* performs well in complex scenarios involving dynamic environments and human robot interaction while providing significant benefits in orchestrating present abilities and integrating new functionalities into the system.

According to the findings of the preceding section, the focus of this thesis is on the development of a control software for *service robots* which is designed to cope with dynamic everyday environments and continuous interaction with the robot's user – even during task execution. The keywords used in the title

**Behavior-based Control for Service Robots inspired by Human Motion Patterns** represent the most prominent characteristics of the control system:

**Service** is the main goal of the control system: Enabling robots to perform services for – and in collaboration with – people. The architecture of the control system enables the user and the behavior modules of the control system itself to simultaneously and with equal priority exert control on the robot's motion. This is in contrast the current norm where the user gives the robot a command and the robot then executes the command ignoring the user until the task is finished (or cancelled).

**Robots** is used consciously: The control system is not tailor made for one specific robot but as independent from the robotic platform and the application as possible. A *Hardware Abstraction Layer (HAL)* renders the control system independent from the specific robotic platforms, and an application-specific layer from the individual applications. This includes for example an application logic and sets of commands for a communication or dialogue system. To enable the re-use of the central layers of the control architecture, the individual layers have to have defined and invariant interfaces. Standardization and modularization play an important role in the roll out of technology and the reduction of development costs.

**Human Motion Patterns** refers to the general application scenario: human everyday environments. The control system enables the robot to operate in environments which are designed by humans for humans. As a matter of course, the ability to operate in such environments does not depend on the control alone, but also on the locomotion abilities of the individual robotic platform. From a control strand point, there are several common challenges in these environments which have to be tackled by the control system, mostly arising from dynamic and cluttered nature of the environment (details can be found in Section 1.3 and 1.4). As these environments are successfully mastered by humans, the control system shall be inspired from motion patterns which can be found when observing humans moving in their environment. These motion patterns are the core component of the control architecture and the *behavior-based control system* which are used to implement these motion patterns.

**Behavior-Based Control** This term stands for a special kind of robot control software which provides an alternative to common monolithic and trajectory-based motion planners. A *behavior-based* control system consists of many independent and self-governed modules following their own goals where the overall behavior of the robot emerges from the fusion of the individual behaviors. This concept supports implementing a multitude of individual behavioral patterns, like those found when observing human behavior. In contrast, forging these into one monolithic system would be extremely difficult. But the behavior coordination is a challenging task and thus *BBCs* are usually only found in low-level applications, if at all.

To curb the scope of the work presented, it should be noted that the goal is the development of the control architecture as well as the fundamental behaviors needed to operate in dynamic environments. These cover amongst other things the collision avoidance with static and moving obstacles as well as the interaction with a user. The special abilities needed for individual applications are not be part of the core control system.

In order to evaluate the control system, the additional hardware- and application specific layers were developed for an exemplary scenario: shopping in a supermarket assisted by a shopping robot. This scenario

was used as recurrent theme throughout the development of the control system. As part of this scenario the robot *InBOT* (*Interactive Behavior-Operated shopping Trolley*) was developed.

### 1.3. Recurring theme: scenario supermarket

The shopping assistant robot in the supermarket scenario addresses several everyday problems which will be analyzed more closely in a survey described later on in this chapter. These are, for example helping the customer to find desired products without extensive search or relieving the customer from the constant burden of pushing the shopping cart using his own force. The latter is especially beneficial when the cart is heavily loaded or the customer is elderly or handicapped. It is mandatory that the robot can maneuver safely in cluttered corridors and is able to find efficient paths to desired products.

Supermarkets are chosen as the main theme in this thesis because they pose a challenging scenario. They are a very dynamic environment and usually have a cluttered and nested character (shown e.g. in Fig. 1.9). This includes the presence of cluttered corridors, moving objects and the fact that the spatial arrangement of the environment changes frequently: advertisements, special offers, or bargain bins are placed or removed daily (see Figures 1.9 and 1.10). This is a great challenge to mobile robots as this results in not being able to rely on a global metrical map unless it is updated constantly, even while the shop is open and customers are around. As a result, a control architecture is needed which enables the robot to navigate without a global map and to localize without relying on the very popular map-matching methods. The robot has to avoid collisions with static obstacles and on top of this is confronted with moving obstacles. These could for instance be customers with shopping carts who are hurrying down a corridor while being distracted by the products in the shelves or talking with each other. The avoidance of accidental collisions with these moving obstacles is essential as even physically harmless collisions could result in psychical damage: people could become afraid of the robot – a worst-case scenario for *service robots*.

In the course of this thesis the supermarket scenario is used to implement the core control system in a specific and demanding application where it can be evaluated. To enable the evaluation, the shopping robot *InBOT* (*Interactive Behavior-Operated shopping Trolley*) was developed. Figure 1.11 shows *InBOT* and the enhanced conventional shopping trolley *ETrolley* which represents a passive shopping assistant. The robot *InBOT*, containing the developed control system, is finally used to perform a user study in a small simulated shop in the laboratory.

The next section will elaborate on the challenges posed by the chosen scenario and on resulting requirements. More details on *InBOT* and *ETrolley* can be found in Appendix B “The Interactive Behavior-Operated Trolley (*InBOT*), *InBOT-2*, and *ETrolley*”.

#### 1.3.1. Infrastructure of the supermarket

The supermarket scenario offers a very wide range of possible infrastructures. As the infrastructure is not part of the scientific concept of this thesis, this section will just name constraints and assumptions made for the overall infrastructures without claiming completeness.

It is assumed that a mixture of different shopping carts is used, including full scale shopping robots but also carts extended with displays and just ordinary shopping carts. The carts which include IT components communicate both, with a central server which maintains the the overall system and de-centralized with other carts or robots. The central server provides a *topologic-metrical map* of the store as well as a database



Fig. 1.9.: Shops contain several objects which are moved on a daily basis, outdated global maps



Fig. 1.10.: Removable special offer bin mounted on top of a pallet

containing the products along with their location. Additionally, the server can be used to observe and steer the traffic by modifying or annotating the map. The de-central communication is used for the coordination of the robots among each other. This way multi-robot behavior is enabled such as avoiding collisions, preventing deadlocks or following each other. In their vicinity they share their *local world model* primarily including their location, goal and current driving direction and velocity.

The shop is outfitted with landmarks for the global self-localization. These may be sparsely distributed, but have to enable the robot to identify transitions between *topological areas*. In this thesis barriers consisting of ground mounted RFID transponders are used, but these can be substituted by any other system which fulfills the requirements – mainly the robustness versus occlusion.

Environmental sensors as described in Appendix C.3 “The *intelligent environment*” are supported by the concept, but optional. The control system depends on dependable information on the user and moving obstacles, but the source of this information is not constrained. In this thesis two different possibilities are presented, one with only on-board sensors and one using an intelligent environment.

Concepts for the continuous operation such as charging strategies and low-maintenance operation are not covered here. They consist mainly of hardware components, task planning, and most important the robustness of every single component and their integration. Including this topic would have posed an additional major effort while not contributing to the scientific concept of this thesis – even though the topic of robust continuous operation is very challenging and systems capable of this deserve highest respect.

### 1.3.2. Challenges posed by the supermarket scenario

The supermarket scenario poses several challenges which have to be solved by the robot’s control system. Obviously, there are additional challenges when taking individual sensor or perception systems into account. For example a speech recognizer will have difficulty identifying the voice of the user amongst the many background voices. Or very colorful shelves are a unfavorable background when considering optical obstacle detection. The following challenges are described from the point of view of the control system:

**Size and shape:** The robot shall have the size needed to carry goods. In the chosen scenario this is the volume as well as the mass of a large shopping. As a result, the robot cannot be built in the beneficial slender and circular shape which is often found when looking at mobile robots (compare Fig. 1.11 with





Fig. 1.11.: The shopping robot *InBOT* and the passive shopping assistant *ETrolley*

Figures 1.6 to Fig. 1.8). Hence the control system has to be able to cope with the large and rectangular shape of the robot, especially when maneuvering in narrow spaces. This includes the necessity of additional safety behavior to supervise the robot when turning which would not be necessary with a round shape.

**Holonomic drive:** In order to be able to operate in narrow and cluttered spaces a robot of the described size and shape needs an holonomic drive system – just as ordinary shopping carts have a holonomic chassis (see also Figures 1.12 and 1.13). Hence, ...

- The system has to control one additional degree of freedom
- The hazard area of the robot is not only in front of and behind the robot as it would be when using a differential drive. In the case of a holonomic drive the robot can simultaneously and independently accelerate in the X- and Y-directions. Hence the hazard area is a full 360 degrees which has to be taken into account when designing the safety behaviors.
- Calculating a precise odometry for an holonomic drive is much more difficult compared to differential drives, even dedicated hardware might be necessary.

**Navigation:** Supermarkets usually contain large quantities of obstacles and mobile – or at least not necessarily static – objects. Which means that ...

- Several static obstacles impair the robot's navigation. Especially large quantities of parked shopping carts can construct large obstacles of various shapes.
- Several moving obstacles have to be taken into account: customers are walking around, some of them pushing shopping carts. They will probably be distracted either by searching for goods or by talking to other customers. These are the majority of moving obstacles. But there are also smaller ones such as dogs or children as well as larger ones like staff operating forklifts.

**Mapping and global self-localisation:** The large amount of non-static objects, whether these are parked shopping carts, advertisements or bargain bins, result in a continuous alteration of the local environment. In fact the amount of positively static structures is rather low and the individual structures look very similar. Hence global maps will be outdated very quickly. This means:

- That these maps are only of limited use for global self-localisation.
- That these maps are only of limited use for navigation and route planning.
- That the maps would have to be updated continuously during the opening hours of the shop.

**Interaction:** All tasks in this scenario have to be performed together with an inexperienced user

### 1.3.3. Survey with customers at the supermarkets

After having identified the challenges posed by the environment in the scenario, the focus is shifted towards the requirements of the potential users of the *service robot*. To be able to assess the expectations of potential users of the shopping robot, a survey was conducted ([Bre09]). There were 187 participants at two supermarkets and 113 participants in an online-survey. Below, only the surveys at the supermarkets will be taken into account as the online-survey shows a strong distortion in age- and gender-distribution. The

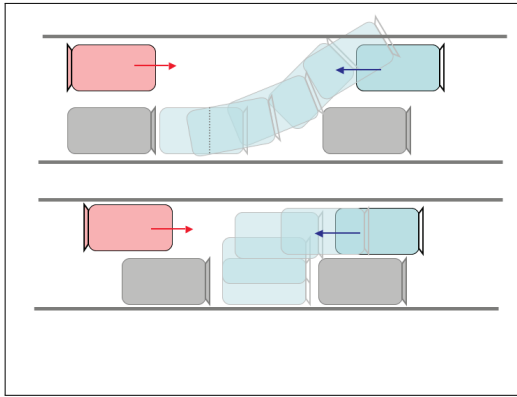


Fig. 1.12.: Comparison: parking the robot between two shopping carts with and without holonomic drive

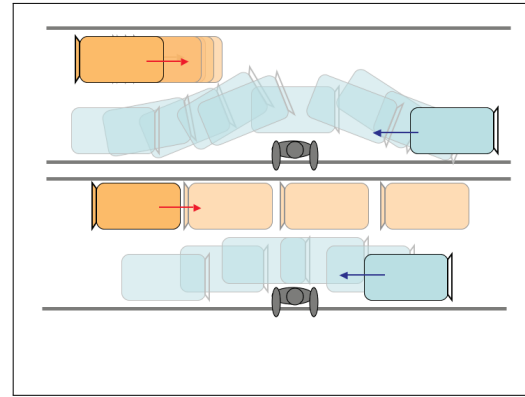


Fig. 1.13.: Comparison: dodging an approaching shopping cart with and without holonomic drive

surplus on young, male students would alter the results in favor of people who are not in need of assistance and who are interested in technology.

In the survey at the supermarkets the age distribution roughly follows a normal distribution with an average age of round about 40 years. The participants were divided in six age-groups: <18, 18-31, 32-45, 46-59, 60-73, and above 73 years of age. The first and the last group are statistically of limited relevance due to the small number of participants (3 in the first and 8 in the last group). The gender distribution slightly favors male with 54%. A summary can be found in Table 1.1.

When interpreting the results, one should focus on an realistic goal for the application: in the foreseeable future one will not see hundreds of shopping robots in supermarkets. It will most probably start with only a few as a PR campaign and maybe reach a preliminary saddle point at round about a dozen robots, dedicated for customers in need of assistance. Therefore the shopping robots should be designed for the few people who really need assistance in their daily shopping or those who are not able to go shopping without assistance any more. Thus the input of the age-group above 73 years is of interest even though it is of limited statistical significance. Finally one should keep in mind that shop owners might follow a different opinion, favoring a marketing campaign or enhancing the *shopping experience* instead of assisting those in need.

**Shopping behavior:** For the general understanding, it is important to know that most of the participants seldom visit other shops than their favorite one (only 17% go to “foreign” shops on a regular basis) and thus know their shop very well. About 70% stated that they usually know where to find their products. Only the age-group of above 73 is an outlier with only 37%. But, even though knowing their shop very well only 52% denied having to search for items from time to time. In the two oldest age-groups more then one third stated that they often have to search for products. Therefore, in these two groups 34% and 62% respectively stated that they would like to have more assistance when searching for items. A phenomenon which should be kept in mind are the technophile: in the younger half of the age-range a much higher rate of the people stated that they would like to have a technical shopping assistant compared to the older half of the groups – even though the younger ones wish less help in general.

Regarding the general shopping behavior about 35% of the participants stated that they use a shopping list very often and about 65% that they use a list at least on a regular basis. In the older groups these figures rise to 50% and 70% respectively.

Tab. 1.1.: Survey: summary and general information

187 + 113	participants in the two surveys (interview + online)
About 40	is the average age; age distribution follows normal distribution
54% male	– almost equal number of both genders
About 70%	usually know where to find the desired products
About 50%	spend time searching for products at least from time to time
About 30%	of the elderly people have to search for their products often
34% to 62%	would like assistance in searching products
65%	use a shopping list at least from time to time

Having to search for products is a common annoyance.  
 Many people use shopping lists from time to time.

Tab. 1.2.: Survey: features wished by customers

62%	Information about special offers
47%	Information about prices
40%	Information about nutrition facts
52%	Cooking recipes available on demand
30%	Not to have to push the cart themselves

No single feature is wished by all customers.  
 Hence, all features have only to be on-demand.

Tab. 1.3.: Survey: designs options

D	2,62
F	1,96
B	1,81
A	1,78
E	1,77
C	1,57

Rating of the design options  
 of Fig. 1.14: higher is better.

**Individual Features:** The participants were asked to directly evaluate a list of possible features of a shopping robot. A summary can be found in Table 1.2. The most prominent rankings received: information on special offers (liked by 62%), information on prices (47%) and information on nutrition facts (40%). About 25% would like the robot to automatically provide cooking recipes, 52% would like to have them available on demand. The feature that people do not have to push the cart using their own force provided for a surprise: about 30% in the younger groups would like this at least from time to time and 15% often but only 20% and 5% in the elderly groups. When asking the elderly customers about this feature afterwards, many of them reacted almost insulted and said something in the line of “I am still able to do this myself!”. It seems that questions regarding this feature have been perceived more as a question of honor rather than a question of comfort.

All these figures illustrate one common fact: there is no individual feature which is wished for by a large majority, but many people would like assistance features. This results in an important demand: one must be able to personalize the robot and the features are only to be provided on demand, rather than automatically.

**Qualitative Information:** Besides these quantitative figures there has also been interesting qualitative information from individual participants. Not enough people shared or mentioned these opinions to be statistically significant but these are interesting for an assistance system nevertheless. For example some women have been shopping with prams. Some of them complained that they cannot do a large shopping because of the limited space in their pram. Some were even putting items on top of the child. Some of them

mentioned that a shopping robot would be nice when it would just follow them so that they could dump their goods into it while walking down the corridors.

Summarizing we have the following aspects which the shopping robot should fulfill from a customers point of view:

1. **Product information (price, special offers, nutrition facts)**
2. **Shopping list management**
3. **Guiding robot** helps searching for products
4. **Cooking recipes**
5. **Features only on demand**
6. **Following robot** lets one have the hands free e.g. to push a pram

When trying to group these aspects one discovers that most of them are in the field of the user interface or more precisely a graphical user interface on a screen. But also two general modes of operation are suggested.

7. **Easy to use graphical user interface**
8. **Guiding Mode**
9. **Following Mode**

**Design of the shopping assistant:** Finally, the participants were shown six drawings for possible designs of a shopping robot. They had to rate them according to how much they liked each individual design. The individual designs are shown in Figure 1.14. The results are summarized in Table 1.3. The participants acted rather conservatively: the design resembling an ordinary shopping cart was preferred compared to humanoid or fancy designs.

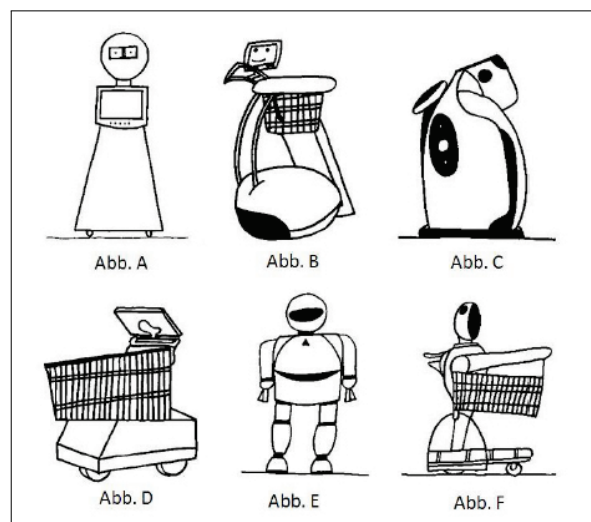


Fig. 1.14.: Design concepts for the robot shopping cart [Bre09] (see rating in Table 1.3).

#### 1.4. High-level requirements for the control system

By choosing supermarkets as the main theme we have a very complex and challenging scenario as was described in the preceding sections. Presently most *service robot* systems force the human to adapt to them. During the design of the control system the focus will shift back to the main idea of *service robots*: adapting the robots to the people. The main topics to be addressed here are coping with the dynamic human everyday environments as well as the multimodal interaction with the user. Here robustness, reliability, and predictability are an important goal. While a trained operator who uses a robot to earn his living will get used to little annoyances, a new user who wants to use the robot as assistant will be repelled. And of course, collision avoidance has highest priority as there are people around the robot which could be hurt. From the challenges described earlier a set of high-level requirements for the control system arise. These are elaborated on below:

**R1 Assistance features:** The robot shall provide the nine assistance features identified in the survey.

**R2 Human-Robot Interaction capability:** As mentioned, the core control system shall enable the robot not only to accept and execute commands in an sequential order but to take the user's actions into account at all times even while executing given tasks. Of course, in addition to this ability, the application-specific layer of the control system needs an easy to use multimodal user interface. As the topic of user interaction is a focus of this thesis the corresponding requirements will be further elaborated in the sections following this summary.

**R3 Abstract and reliable environmental representation:** The environment is altered continuously. The rate of these alterations can be too high to be able to maintain a global metrical map. Therefore, the control system has either to be able to cope with an incorrect map or the global map has to be that abstract that it is not effected by the alteration as for example a *topological map* could be.

**R4 Self-localization:** A robust self-localization concept for a holonomic drive has to be developed consisting of both hard- and software which provides for a suitably precise odometry as well as the ability to perform global positioning corrections based on landmarks. This system must be able to detected the landmarks reliably even in highly frequented areas with many occlusions. As argued earlier, the control system shall not rely on the popular map-matching methods as these rely on an accurate and up-to-date global map, which cannot be guaranteed in this scenario.

**R5 Robust navigation:** The control system has to be able to avoid static and moving obstacles while staying target oriented. It has to be able to generate a path to a given goal on a high level using an abstract map as well as on a low level only relying on current sensor data. This way the robot can on the one hand generate efficient long term paths and on the other hand has suitably low reaction times to sudden environmental changes. Additionally, fail-save basic behaviors are needed which can compensate failures of modules for environmental perception and at least render the robot in a stable state.

**R6 Safety:** For operating with untrained users the safety of the system is crucial. The user (and other people) must not be harmed. But, even if no danger for the people is apparent they might feel scared by the robots movements or little (actually harmless) bumps and thus cease using the robot. Accordingly,

the control system has to ensure that the robot will not (actively) run into any obstacle and it shall try to avoid collisions resulting from moving obstacles which would otherwise run into the robot. The second point cannot be made a hard requirement as the robot will have limited velocity and acceleration capabilities and thus the control system cannot guarantee to always be able to evade approaching objects.

**R7 Multi-Robot-Interaction capability:** Finally, thinking of a fleet of robots operating in one supermarket these robots have to cooperate. Multi-robot collision avoidance is needed, the robots need to be able to queue up, for example at self-service counters or check out counters. And it would be useful if one user could operate several robots simultaneously, be it for maintenance or because a customer wants to buy large amounts of products. Obviously, this requirement is less demanding compared to multi-robot collaboration scenarios where for example several robots have to manipulate a large object together.

**R8 Generalization:** It shall be possible to apply the control system on different *service robots* in different applications. More details will be elaborated in one of the following sections.

#### 1.4.1. R2 Human-Robot Interaction capability: user interaction

The requirements regarding the user-interaction capabilities can be divided into three groups or sub-requirements, which will be defined below. T.B. Sheridan has published high impact work in the area of human factors in robot control, thus, definitions made by him will be taken up here ([147], details will be given in Chap. 3.1.4).

**R2.1 User interface for commanding:** This is the traditional approach. The user gives a command to the robot and the robot executes the command afterwards. T.B. Sheridan named this *control trading*. The requirements for this topic center around designing an intuitive user interface consisting of several modalities. This clearly belongs to the application-specific layers of the control system. The robot *InBOT* for example merges a touch screen-based GUI, a headset for speech in- and output and a bar code scanner into one integrated user interface.

**R2.2 Control sharing:** Having a dedicated user with whom the robot has to cooperate, a focus is put on a challenge commonly found in collaborative activities: sharing of the control. Using this approach the robot shall be able to take the users actions into account while executing a task so that the user does not have to consider all eventualities before giving a command. T.B. Sheridan named this *control sharing*. As the potential tasks differ in complexity and hence control-layers of varying abstraction are active, the control system has to be able to accept control input by the user in all abstraction layers of the control system. Often the user exerts his control share unknowingly. Examples could be the relative position of the user, the estimated goal, the motion or even the acceleration of the user, the user's gaze direction, but also a force which the user exerts on a force sensor.

**R2.3 Modes of operation:** This is a more abstract group which utilizes the abilities provided by applying *control trading* and/or *control sharing*. The control system has to be able to operate the robot in different ways, depending on the users wishes which he expresses using a set of commands. These different ways of operation can be modeled by defining individual *modes of operation*. These are described in the next section.

### **Modes of operation for user interaction (R2.3)**

The control system shall be designed in a way that the user can easily adapt the robot's behavior to the user's specific desires. In the context of the supermarket scenario the survey described earlier identified two fundamental ways in which potential users wish to operate the robot: The first group wishes to shop just as normal but wants to get rid of pushing the shopping cart and the second group wants the robot to guide them to products. These two fundamental ways of using the robot result in three *modes of operation*: (1) The *Following Mode* for those who just want to get rid of pushing a shopping cart and a combination of (2) the *Guiding Mode* and (3) the *Servoing Mode* for those who want to be guided to products. A third group of users wants the robot just to be an information kiosk which provides directions or product information. As this is not a way of operating the robot this is not mapped on a *modes of operation* – the user can use the robot's UI to get information in all modes. Here follows a list of the defined active modes:

1. **Autonomous Mode:** After receiving a command the robot executes it autonomously. This mode is used for local interactions (e.g. “come here”), to temporarily get rid of the robot (e.g. “wait at <location>”) or for maintenance tasks commanded by a central server or the shop's personnel. This mode uses the traditional *control trading* way.
2. **Guiding Mode:** The robot guides the user to a given target location e.g a product or the checkout counter. The control system has to observe the user's movements to be able to slow down or wait for the user if he falls behind.
3. **Following Mode:** The robot follows the user in a defined distance and navigates around obstacles if the user chooses a path which the robot cannot follow.
4. **Servoing Mode:** The robot imitates the movement of the user (as good as the environment allows). This mode is the successor of the *Guiding Mode*. After guiding the user to a shelf the user starts moving along it to analyse the offers. The robot will follow or will be virtually pushed along in front of the user so that the robot will not impair the user. Again the robot navigates around obstacles autonomously.
5. **Manual Steering Mode:** Here the user steers the robot directly using a force-sensitive input device. The control system assists the user by avoiding obstacles on its own – as long as the user does not overrule the control system. The user is even able to give local commands like “park on the left side” to the robot by applying *Force Commands* therefore by exerting sharp forces on the device. The robot then switches to *Autonomous Mode* and executes them.

These active modes are accompanied by two passive modes: the *Idle Mode* and the *Waiting for Login Mode*. In both cases the robot just stands still and waits either for a user to login or for commands by its user. While doing so the robot shall try to avoid collisions caused by moving obstacles by evading them.

**Mode transitions:** As illustrated in Table 1.4, all modes utilize a subset of the modalities available. If the user gives a command via a modality that is not coupled with the mode the robot currently operates in, a mode transition is performed. For example if the robot operates in the *Manual Steering Mode* and a voice command is received the robot automatically switches to a mode which is coupled with voice commands.



Tab. 1.4.: Dependencies of modes and modalities of interaction.

Mode	Force sensitive dev.	User position	Gesture recognition	Voice commands	Touch screen cmds.	Voice out	Screen out
Manual Steering	X	-	-	-	-	X	X
Servoing	-	X	-	-	-	X	X
Following	-	X	X	X	X	X	X
Guiding	-	X	X	X	X	X	X
Autonomous	X*	-	X	X	X	X	X
Idle	-	X <sup>+</sup>	-	-	-	X	X

\*: In the *Autonomous Mode* the handle is used to give force-based commands only, not to steer the robot.

+: If the user is lost the robot switches to the *Idle Mode*.

This could for example be the *Following Mode* if the command was “follow me” or the *Guiding Mode* if the command was “guide me to [name of product]”.

The transition from and into the *Manual Steering Mode* is also triggered by an explicit command – but in contrast to the other modes by a non-verbal one: *InBOT* automatically switches to the *Manual Steering Mode* as soon as a force is applied to the handle and switches to *idle* when the handle is released or to *autonomous* when a *Force Command* is detected.

The user position tracking does not transmit direct commands from the user to the robot. It passively commands the robot to accelerate or decelerate and therefore does usually not result in a mode transition. An exception takes place if the user is lost. Then a transition to the *Idle Mode* is performed which lets the robot wait for the user – after a defined amount of time an automatic switch to *autonomous* is possible to enable the robot to search for the user, if wished and if it makes sense in the application. The usual way for the robot to switch to the *Idle Mode* is the completion or the cancellation of the active task. The *Servoing Mode* is also activated automatically: it is enabled when the robot reaches a goal in the *Guiding Mode*. This way the robot accompanies the user at the goal location. The common problem of oscillations between modes does not appear here because mode transitions are always triggered by sparse events such as user commands, reaching of a goal, grabbing the handle, etc.

Therefore the user does not explicitly order the robot to switch to a dedicated mode but the robot performs a mode transition automatically based on the given command and on the modality the command was given by.

### 1.4.2. R8 Generalization

This final group of requirements is not motivated by the specific scenario. In contrast, these shall allow the transfer of the control system developed in the context of the supermarket scenario to other applications. The collected requirements are relevant not only for the supermarket but also for several other indoor ap-

plications like general guiding tasks, transportation or inspection and outdoor applications like exploration or observation. For every application and every target environment a tailor-made robot is needed, outfitted with a corresponding set of sensors, actors and control components. But when the core control system can be separated from the application- and robot-specific requirements it can be re-used on several robots and applications. To achieve this re-usability the core control system must provide a set of defined interfaces for commands from an application logic “from top”, for the sensor processing chain “from the side” and finally for instructing the platform “to the bottom”. Moreover, it would be beneficial when it would be possible to use only certain components of the control system in individual applications when some functionalities are not needed or provided otherwise. Hence, the interfaces have not only to be defined “around” the core components of the control system but even between them. This way a modular and portable control system can be created.

### 1.5. Approach for the *service robot’s* control system

This section describes a rough concept of how the described challenges were tackled and the requirements were fulfilled. Hence, this section provides a short preview of the topics addressed in the remaining chapters of this thesis.

This concept is organized in four groups: the control architecture, the core control system, the application-specific components and the evaluation on real robots including user-studies. These components will be briefly introduced in the next sections. The interplay of the main components is sketched in Figure 1.15.

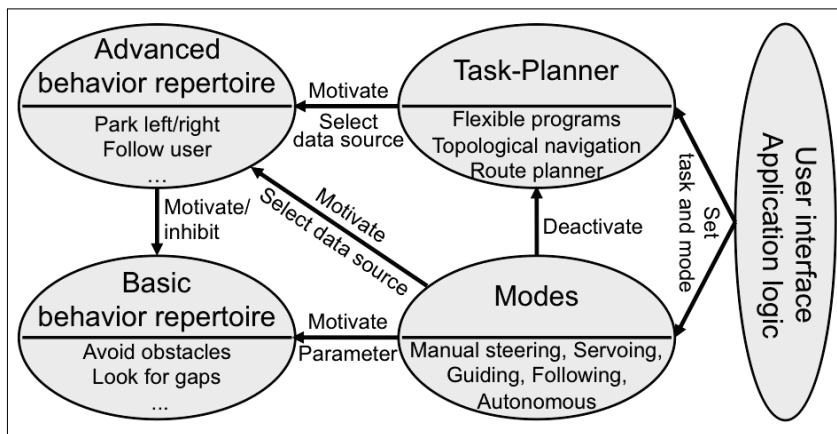


Fig. 1.15.: Interaction of *task planner, behavior repertoires* and *modes of operation* in the control system.

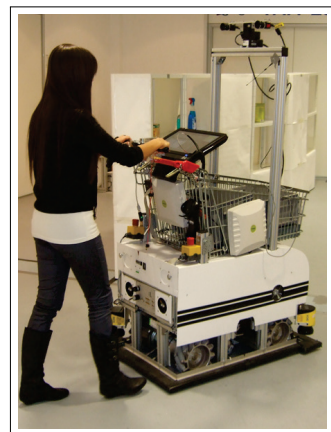


Fig. 1.16.: The shopping robot *InBOT*

#### 1.5.1. Concept of the control architecture

As the navigation in human everyday environments is a very challenging task, an option for designing the control architecture and the control system itself is to let one be inspired by subjects which are already able to operate in such environments perfectly well. Thus, the control architecture is inspired by research on human motion patterns performed by S.P. Hoogendoorn [73] in the context of *Evacuation Dynamics*. This research suggests that the motions patterns of humans can be modeled using a tree-layered architecture ranging from strategic over tactical to reactive / operational behaviors. This model is applied in the control architecture of

this thesis. The control architecture integrates a *topological navigation* in the *strategic layer*, a deliberative *Behavior-Based Control (BBC)* in the *tactical layer* and a reactive *BBC* in the *reactive layer*. The *BBC* is inspired by previous work by J. Albiez [4] who designed *Behavior Networks* for the control of walking machines. The individual layers have defined interfaces to enable two important features: first attaching platform- and application-specific layers and second feeding the orthogonal control input in between the layers to enable *control sharing*.

### 1.5.2. Application-specific part of the control system

As the shopping scenario is used as the recurring theme of this thesis, a corresponding shopping robot was designed: the *Interactive Behavior-Operated shopping Trolley (InBOT)*, Fig. 1.16, see Annex B) which is used for development and evaluation purposes. Additionally, the passive shopping assistant *ETrolley* was constructed as an alternative application for the scenario as well as for user studies. *ETrolley* runs the same control software as *InBOT* does, therefore *ETrolley* is capable of localizing itself and to perform route planning whose results are displayed on the touch screen as navigation assistance.

Three groups of components had to be developed for this application: the **user interface** including the product database and the *force sensitive handle*, the **application logic**, and finally a **self-localization** concept in hard- and software for holonomic drives in highly frequented environments.

**User interface, force sensitive handle and product database:** First of all, a force sensitive input device was developed to enable the user to push the robot just as an ordinary shopping cart. Later on, an user interface was developed for *InBOT* based on QT to fit on a dedicated touch screen PC. It contains a product database with location and nutrition information, navigation assistance, shopping list management, a recipe database and accepts the touch screen, a bar code scanner and speech output as modalities.

**Application logic:** Beneath the user interface there is an application logic which organizes the high-level behavior of the robot e.g. it interprets the commands given by the user, triggers output to the user and activates task planning and execution. It communicates with the highest level of the core control system – the *strategic layer* – using a defined TCP interface.

**Self-localization concept:** Here two challenges have to be considered: (a) the local odometry has to be calculated even though the robot uses a *Mecanum* drive for holonomic movements. Therefore, passive wheels which measure two dimensions of the robot's movements were developed.

And (b), the global self-localization system has to be robust against crowded corridors where people will continuously occlude landmarks and where parked shopping carts will alter the environment compared to a stored map. In this thesis it was decided to place RFID Barriers – stripes consisting of several RFID tags – across the corridors in the close vicinity of crossroads.

### 1.5.3. Evaluation of the control system

Two main aspects shall be proven right by the evaluation: the capability of the control system in the context of the chosen lead-scenario as well as the hardware- and application-independence of the core control components. For the first case, user studies have been performed with the robot *InBOT* in a mock-up supermarket in the laboratory in addition to the component-based evaluation of the control system. For the second

case, the control system was implemented in several robots. In most of the cases only some components or layers of the control system were used – depending of the target scenario of application – demonstrating its modularity. To gain additional insights, for some components alternative solutions to the proprietary ones were integrated. Two experiments with users have been conducted. The evaluation and the results are described in the chapters 4 “BBC: Navigation, Obstacle Avoidance and Safety” and 6 “User Interaction”.

### 1.6. Contribution

In this section the main concepts and strategies are summarized which are meant to enable the robot to cope with the challenges described previously – mainly the dynamic environment. The resulting abilities often span several layers in – or are orthogonal to – the control architecture. Anticipating the main achievements of this thesis we have:

- Development of an application-oriented robotic system which does not depend on a metrical map but only on a topological map and sensor data from the current field of view.
- Design of a control architecture which is inspired by behavioral patterns of human beings.
- Development of a control system for a large, rectangular, and holonomic platform that is able to cope with a dynamic environment, including the proactive avoidance of moving objects, while simultaneously fulfilling given tasks and interacting with a user.
- Applying the concept of *control sharing* between robot and user by means of five *modes of operation*.
- Development of a *behavior-based* control system using “*Behavior Networks*”, which is application-oriented, including closely connected human-robot interaction and multi-robot behaviors.
- Demonstration that *Behavior Networks* facilitate the easy integration and orchestration of functionalities and thus pose a great opportunity for developing service robots.

**Definition of the scope of this thesis:** The focus of this thesis is on the behavior-based control architecture which is implemented by the *Behavior Networks*, the navigation system which is inspired by S.P. Hoogendoorn’s model of pedestrian motion, and the human robot interaction (HRI) – all in the context of the supermarket scenario. Especially the field of HRI will be limited to the components relevant for the application, i.e. implementing the guiding and following behaviors: giving orders to the robot, receiving feedback from the robot, and sharing control during task execution. Interpreting the humans’ intentions, learning and actual collaboration or even human-robot teams will not be considered. Likewise, complementary concepts such as multi-robot behaviors are only considered as far they are necessary to enable the operation of the system in the scenario of application.

### 1.7. Organization of this work

The thesis is organized as follows (see Table 1.5 for a summary):

The present chapter – “Introduction” – introduces the goal of this thesis which is the development of a service robot’s control system by utilizing *Behavior Networks* to facilitate the easy orchestration and integration of capabilities. Then robot-assisted shopping is defined as an exemplary scenario for application.

Tab. 1.5.: Organization of the thesis

1	Introduction
2	State of the Art
3	The Hybrid Control Architecture
4	BBC: Navigation, Obstacle Avoidance and Safety
5	Avoiding Collisions with Moving Objects
6	User Interaction
7	Multi-Robot Coordination
8	Conclusion, Discussion, and Open Issues
A	Interdependent Work, List of Publications, and Student's Theses
B	The Interactive Behavior-Operated Trolley (InBOT), InBOT-2, and ETrolley
C	Complementary System Components

Based on the scenario and a survey at supermarkets, requirements are defined which will be refined in the subsequent chapters. Finally, the approach of this thesis for the control architecture is described and the contribution is summarized.

The second chapter “State of the Art” introduces a model of pedestrian’s motion patterns which provides an important inspiration for the control architecture, summarizes control architectures with a special focus on behavior-based and hybrid architectures, and provides an overview on service robots in general and on shopping robots in particular.

The third chapter “The Hybrid Control Architecture” starts by describing the main sources of inspiration for the control architecture, namely a model of human motions patterns, *Behavior Networks*, and *control sharing*. Afterwards, the design criteria for the control system are defined taking the requirements defined in the introduction into account. This process of requirements engineering leads to the elaboration of the control architecture with the three layers: *strategic layer*, *tactical layer*, and *reactive layer*. A description of interfaces, *behavior repertoires*, the *local world model*, and of the data flows is also included.

The following chapters will focus on the developed behaviors and functionalities. As many functionalities emerge from the interaction of two or even all three layers, there will not be one chapter describing each layer. In contrast, there will be one chapter for each group of functionalities, describing the relevant impact and interaction of the three layers. These four groups are: navigation, handling of moving obstacles, human robot interaction, and multi-robot behaviors.

The fourth chapter “BBC: Navigation, Obstacle Avoidance and Safety” presents the navigation system: it summarizes the functionalities implemented in the three layers and introduces the core navigation system including self-localization and data acquisition for the *local world model*. The major part of the chapter describes the individual behaviors, ranging from safety behaviors via obstacle avoidance up to the geometrical *scene analysis*. The chapter is concluded by presenting tests performed with the navigation system involving “real” users.

Chapter five “Avoiding Collisions with Moving Objects” introduces the three-stage concept to avoid collisions with obstacles which are moving themselves. Two *reactive behaviors* provide fast reaction times and a spatio-temporal planner is in charge of solving complex situations.

The sixth chapter “User Interaction” presents the human-robot interaction capabilities of the control system. The *modes of operation* (autonomous, guiding, following, servoing, and manual steering) are introduced and the concept of *control sharing*, its implementation, and its effects on the robot’s behavior is displayed. The HRI capabilities emerge from the interplay of all components, but there are still some behaviors dedicated to HRI which will be introduced here as well.

The seventh chapter “Multi-Robot Coordination” introduces the architecture and behaviors of in case of a multi-robot application. There are several necessary functionalities when a fleet of robots – even small one – has to share a common environment such as collision and deadlock avoidance, but also cooperative behaviors like building virtual trains or queuing up.

And finally Chapter 8 “Conclusion, Discussion, and Open Issues” concludes this thesis. It summarizes the developed control system and architecture, presents several robot systems on which parts of the control were implemented and discusses open issues and steps needed to bring the system into a real supermarket.

Afterwards, Annex A “Interdependent Work, List of Publications, and Student’s Theses” discusses other work which is linked to this thesis, such as a list of students’ theses which have been conducted in the course of this thesis. Additionally, the list of publications lists parts of this thesis which were published *ex ante*.

Additional information on various topics relevant for this thesis is provided in the technical annexes.

Annex B “The Interactive Behavior-Operated Trolley (InBOT), InBOT-2, and ETrolley” shows the design of the robot *InBOT* itself including the construction of hardware components used on *InBOT*, for example the odometry wheels for the holonomic platform or the *force sensitive handle*. Furthermore, the current status of the design of *InBOT-2* – a smaller platform with a higher level of integration – is described. And finally the passive test platform *ETrolley* is introduced.

Annex C “Complementary System Components” presents components used on *InBOT* which do not belong to the main control system such as the user interfaces or the system for tracking moving objects or the user. Often, two alternative versions are presented, one developed in the course of this thesis and one by a third party to increase the possibilities for evaluation and to show the independence of the core control system.

## 2. State of the Art

This chapter summarizes work related to the utilized or developed technologies of this thesis. It starts with a brief introduction on the analysis of pedestrian's motion patterns (Section 2.1) which provides the basic concept of the control architecture developed in this thesis. Related work regarding control architectures in general can be found in Section 2.2, regarding individual components of the navigation system in Section 2.3 and finally regarding integrated assistance systems and *service robots* in application can be found in Section 2.4 and with special focus on the supermarket scenario in Section 2.4.3.

In order to keep this chapter slim, fundamental concepts from the fields of navigation, planning, or obstacle avoidance will only be introduced here if required to understand the robot systems in Section 2.4 or the control system developed in this thesis.

### 2.1. Brief introduction to pedestrian motion modeling and potential applications in robotics

Analyzing pedestrian's movements is not a new field of research: At the end of the 19th century several disasters motivated the research. For example in December 1881 the theatre in Vienna burned down, killing hundreds of attendees and enforcing the research in pedestrian's dynamics [39]. The main focus was placed upon evacuations to be able to save human lives [44]. Today, this is still an ongoing and relevant topic (e.g. [138]) thinking of even worse firetraps like tunnels or subway stations.

Beginning in midst of the 20th century researchers began modeling and simulating traffic, first focused on streams of cars, and later on streams of pedestrians as well (e.g. [58]). The economical benefit was driving the research: by simulating the traffic the design of crossroads or subway stations could be outfitted with an optimal tradeoff between size and the necessary capacity, saving the cities and companies from expensive failures.

Up to today scientists have collected a huge amount of knowledge, tools and methods. What can robotic engineers learn from them? Some time ago robots were developed mainly for industrial applications or as research platforms to operate in labs, screened from the public. But today the robotic society pushes into public spaces and designs service robots for specific public applications like museum guides or shopping assistants. Here the robots have to operate in the same space like people without confusing or disturbing them. The knowledge of human movements can be used either to enable the robot to guess the behavior of nearby people or to behave in a human-like manner itself. To make the knowledge applicable, a corresponding model is needed. A popular model was developed by J.P. Hoogendoorn [73] to simulate the motion of groups of pedestrians. It will be briefly introduced here. In the chapter discussing the design of the control architecture (Chapter 3.1.1) its exploitation as source of inspiration for the control architecture will be illustrated.

Hoogendoorn divided the behavior of simulated pedestrian agents in three hierarchical layers (as illustrated in Fig. 2.1).

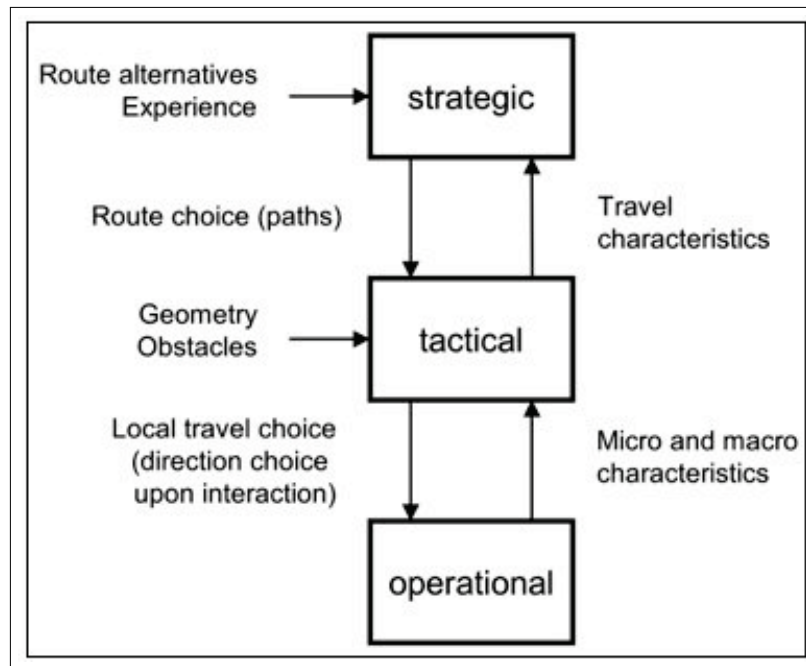


Fig. 2.1.: Hoogendoorn's Pedestrian Behavior Hierarchy [73]

1. *Strategic behaviors*: The topmost layer represents the human route selection based on landmarks. It defines roughly the way that has to be taken to move from the starting point to the goal. A list of nodes containing landmarks or areas that have to be passed/crossed is generated here instead of a continuous path. In pedestrian movement simulation the set of possible nodes is defined by the scenario designer manually.
2. *Tactical behaviors*: The middle layer refines the given (topological) route based on the local environment's geometry such as infrastructure like stairs, doors, walls or corners. Here again nodes are generated instead of a continuous path. Connecting the individual nodes gives a edge-mesh of paths in the local environment. Pedestrians are supposed to choose a shortest path in this mesh. Based on the experiences with corner-oriented cell-based methods introduced by T.Kretz [97], M. Bluemel modeled the mesh as corner-based visibility graph because it does not contain dispensable nodes and always contains the shortest path [19]. When comparing this model to paths actually taken by people, it seems to fit well while still matching Hoogendoorn's description of the hierarchy. Figure 2.2 shows some examples.
3. *Operative behaviors*: In the bottom part of the hierarchy the real movement behavior is generated. Here the agent moves from node to node until it reaches the target destination. While doing so, it adjusts the path to local disturbances like obstacles or other agents. In simulation this is often done using local potential fields.

Within these three layers three groups of behaviors identified when observing pedestrians are incorporated:

**Route selection of pedestrians:** According to A. Millonig [115], the decision of a pedestrian for a certain route depends on five criteria: *attractiveness, availability, infrastructure, safety, complexity, and*



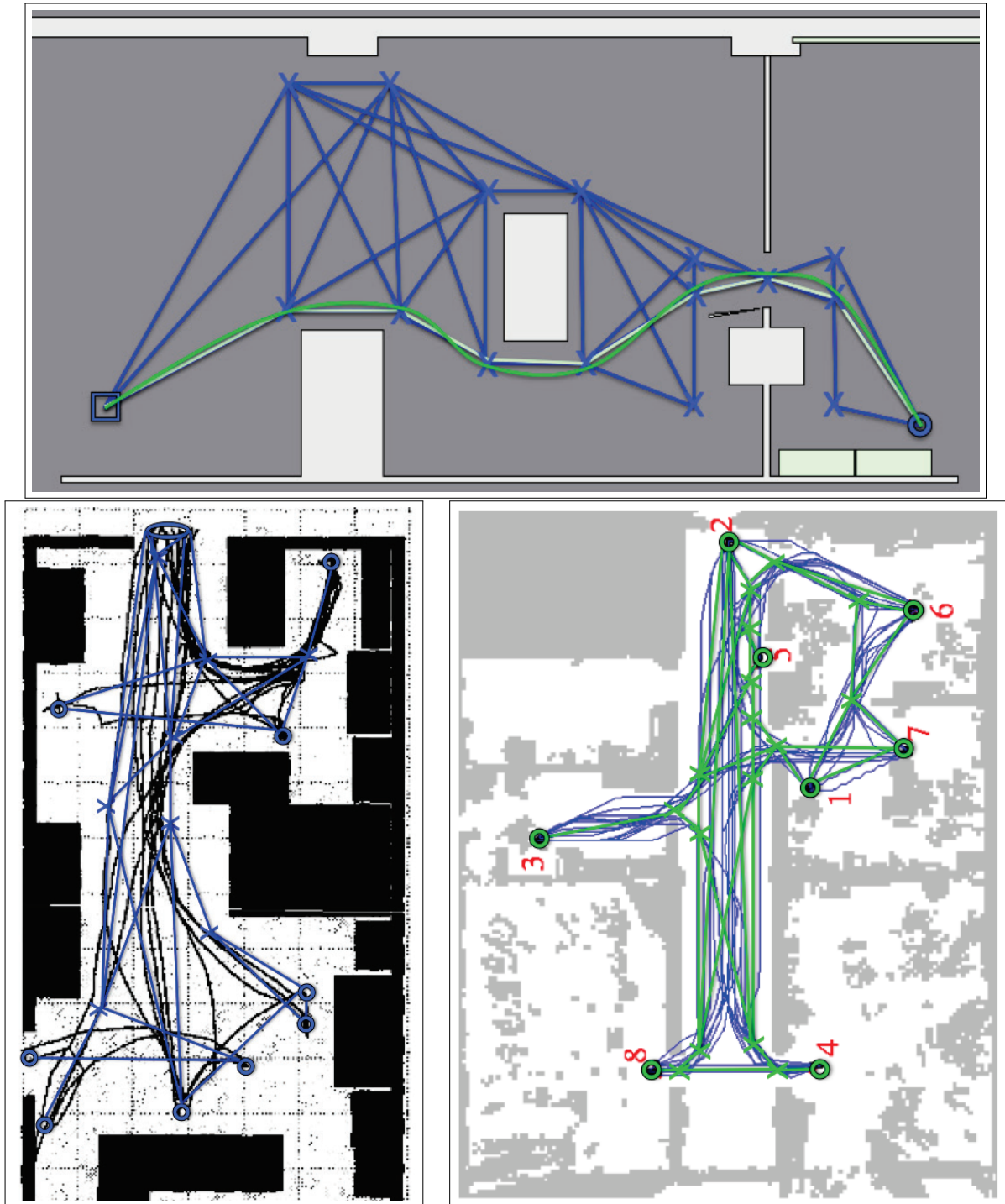


Fig. 2.2.: Introduction of edge-meshes as motion model of pedestrians. Top: Edge-mesh of a section of the FZI labs in form of a visibility graph following the concept of M. Bluemel [19]. The light blue line shows the chosen path through the mesh between two hot spots, the green curve an exemplary motion. Bottom: The edge mesh compared to tracked pedestrians' paths. The underlying figures and paths originate from E. Kruse [99] (left) and M. Bennewitz [13] (right).

*topography*. The exact order of priority of these criteria differs for each individual but in general people prefer short and fast routes (*topography*) and those which offer a varied environment (*attractiveness*). This knowledge is of special interest for guiding robots. They can take these criteria into account when generating a suitable route to a given target under the assumption that they can estimate their user's preference. Or the robot could at least be able to inform the user why it has taken a route the user would not expect.

**Local movement of pedestrians:** After the pedestrian has - intuitively - decided for one route, he starts moving along it node by node or landmark by landmark, respectively. The local movements are influenced by two main factors: the desire to reach the next way-point and the reluctance to come too close to any objects, especially other people. Here the *social distance* becomes relevant which individuals try to keep between each other as good as possible. People generally dislike any violation of this distance so *service robots* should try to keep it as well. Keeping this in mind, the robot would know how fast and how close it may approach people and that it must not move past them closely behind their back. If necessary, the robot would at least know when to warn bystanders of its approach.

**Movement velocity of pedestrians:** Additional to the actual path, the velocity of the pedestrian's movement is a crucial characteristic. Therefore the fundamental diagram (Fig.2.3) is introduced by U. Weidman in 1993 (see [177], and evaluations by A. Seyfried in 2005 [146]). It correlates the average velocity with the amount of people per square meter, or the other way around: the rate of free room along the path. A service robot, especially guiding robots, should adapt to this velocity. Otherwise, it would either scare or at least annoy people.

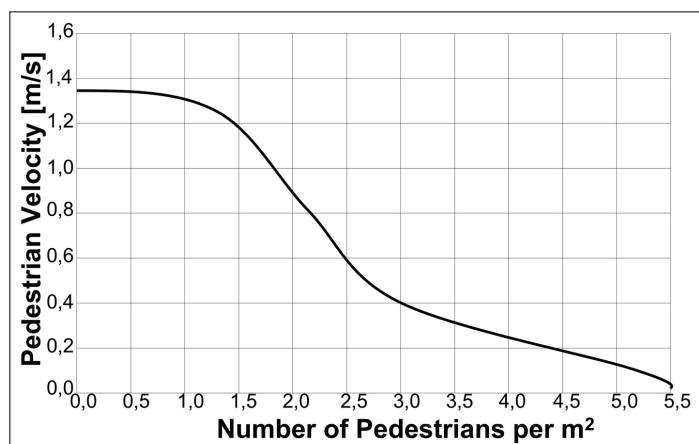


Fig. 2.3.: Example: Fundamental Diagram (FD), in dependence on U. Weidmann's FD ([177])

**Discussion** The knowledge that can be gained from a model of pedestrians' movement patterns can be used either to adapt the robot's movements accordingly or to implement a mixed-initiative communication system: to enable the robot to take over the initiative and inform the robot's user or bystanders about the robot's actions resulting from occurring problems if the robot can assume that these problems are not apparent for the people. While analyzing the work of Hoogendoorn on the behavior hierarchy and the related

research by Millonig, Weidmann, and others, one comes to the conclusion that there are three major cases in which modeling of human movements can be beneficial for the behavior of service robots.

1. *Movement of the user:* Using a model of the estimated movement of the user, the robot is able to predict the user's movement. If the robot is guiding the user, the robot can estimate if the path it intends to take differs from the path the user would prefer. Here the robot can either adapt to the (probable) user preference or it can take the communicative initiative and inform the user about the following unexpected movement.
2. *Movements of bystanders:* If the robot possesses a suitable model of human movements it can estimate their future movements. This can improve the robots' control by estimating the future movement of individual humans nearby enabling the robot to plan a path which will not penetrate the humans' social distance. Furthermore, the movements of whole groups of humans can be estimated which would enable the robot to avoid future crowded areas.
3. *Re-planning:* For the robot there is some kind of fixed – maybe implicitly given – threshold which defines if the robot interprets a situation as traffic jam and therefore re-plans its path. But for humans the borderline is fuzzy and varying from person to person and even current temper. As result, the robot might re-plan its path due to the traffic jam while the user does not recognize the jam at all. Here the robot needs to estimate if the user has detected the traffic jam himself. If this is not sure the robot has to inform the user about the reason for the re-planning. Figure 2.4 shows a matlab simulation of a pedestrian with different crowd densities and Figure 2.5 sketches the estimated density of a crowd and the tolerance of an exemplary human and an exemplary robot.

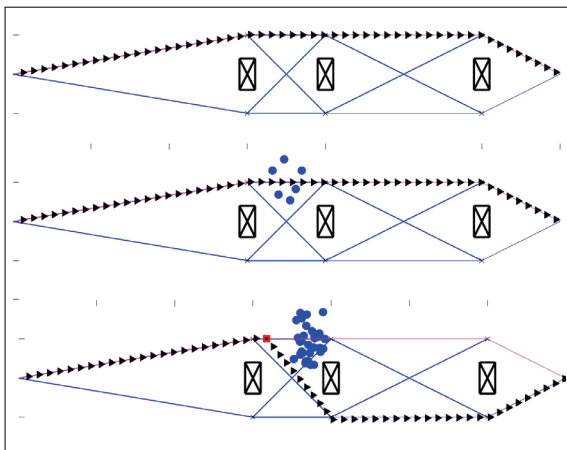


Fig. 2.4.: Simulation of traffic jams with crowds of people with three different densities: (1) without crowd, (2) with a sparse crowd where a pedestrian or robot could pass through, and (3) a dense crowd which has to be skirted (M. Bluemel [19]). Blue dots show the members of the crowd, the triangles the path taken by the simulated agent from the starting point on the left to the goal on the right. The boxes illustrate three obstacles.

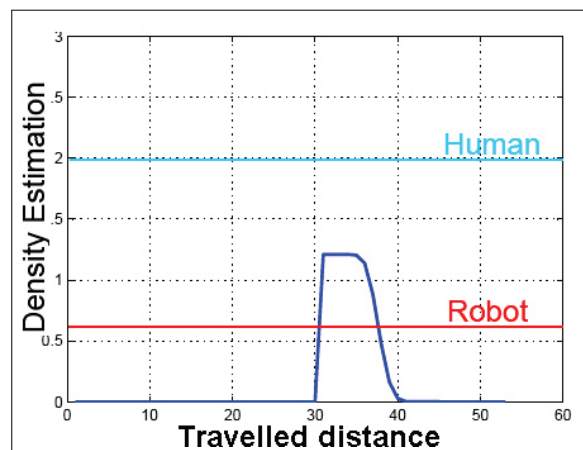


Fig. 2.5.: Density tolerance: the blue line shows the density of the crowd according to the figures on the left, the two other lines the tolerance of human and robot. The robot has to skirt less dense crowds as the human would have to. (diagram based on density diagram from M. Bluemel [19])

## 2.2. Control architectures for mobile robots

This second section of the state of the art shall introduce a variety of control architectures found when analyzing mobile robots. In 1997 M. Mataric [107] analyzed several control architectures. He defined that “an architecture provides a set of principles for organizing control systems. In addition to supplying structure, it imposes constraints on the way problems can be solved.” Thus, the control architecture of a robot is the crucial design element and therefore has to be properly defined. Mataric distinguished control architectures between being *deliberative*, *reactive*, *behavior-based* or *hybrid*. The next four sections will address these four classes and explain their differences and similarities. As this thesis will propose a behavior-based approach, the focus will be on behavior-based and hybrid architectures.

### 2.2.1. Deliberative or functional architectures

These architectures follow the classical planner-based approach. There is no definite term, Mataric [107] for example called them *deliberative architecture*, Ishiguro [82] called them *functional*. Nowadays, the term *deliberative* does not perfectly fit as distinction to *Behavior-Based Controls*, as modern *BBCs* actually include deliberative behaviors or even model planning processes using multiple behaviors [125] instead of using dedicated planners. Accordingly, the term *functional* would be the better match. But as the deliberativity is the most intuitive characteristic compared to *reactive* systems, both terms coexist legitimately.

These architectures have in common that they strongly rely on a complete, up-to-date, and accurate world model, including a model of the capabilities of the robot to change the world. Errors in sensing or changes in the environment which did not happen on purpose force a review of the world model and in consequence a re-planning. In complex real-world environments, especially in the presence of other actors like humans, this can result in significant delays in task execution. And worse, this makes timely reactions in critical situations hardly possible. But besides these drawbacks, these architectures generate plans which either guarantee a correct task execution or at least detect that a task cannot be executed. Especially in complex interlaced tasks the planner-based approach is superior. Consequently the *hybrid* and / or *hierarchical* control architectures were developed which combine *reactive* components with the capabilities of planners (see Section 2.2.4). The main task of the *reactive* components is to ensure timely reactions and to shield the planner from the complex real world environment, letting it only operate on abstract information.

### 2.2.2. Reactive architectures

These architecture follow a strict “stimulus - reaction - scheme”. They do not use a world model nor do they perform planning, the control strategy is programmed into a set of condition-action pairs. Thus a given task is modeled implicitly into the control system. This task cannot be changed (the parameters, e.g. certain coordinates, of the task can) without changing the control system itself. Consequently, reactive systems are limited to tasks which are completely specified at design-time. As the individual algorithms are small, they guarantee fast reaction times, a high degree of robustness and are very predictable. There are several methods of designing the condition-action pairs of reactive systems: lookup tables, rules, circuits, or vector fields. As no world model is kept, these methods rely on the paradigm that the real world is the best model – as far as the used sensor systems are suitable. The reactive systems have proven to be efficient in dynamic and complex environments, but lack the flexibility for more complex tasks which go beyond

direct and straightforward behavior. Here again the concept of the *hybrid* architectures has to be mentioned, combining the advantages of plan-based and reactive systems (see Section 2.2.4).

Mataric [107] discusses where to draw the line between reactive and deliberative (functional) architectures. He states the the main criterion is the amount of computation performed at run-time. Accordingly, a planner which would calculate all possible plans offline and stores them to be looked up at run-time based on current information would be a *reactive architecture*. Would the planner calculate the same plans based on current information at runtime instead, it would be a *deliberative architecture*. This is taken to the extreme by Schoppers [145] in 1987 who aimed at making reactive control systems more suitable for complex tasks. He introduced the idea of automatically compiling a complete control system into a huge set of reactive rules called “Universal Plan” instead of programming the control system manually. This concept would make all currently deliberative control systems reactive and capable of real-time reactions, but it proved to be too poorly scale-able for complex real-world applications.

### 2.2.3. Behavior-based architectures

*Behavior-based architectures* have their origin as (partially biologically inspired) branch of the reactive architectures. But they diverged from their origin quickly. *Behavior-Based Controls (BBCs)* consist of several modules called “behaviors”. These behaviors work independently from each other and all follow their own task or goal. The overall behavior of the robot emerges from the sum of the individual behaviors. The behaviors can be implemented using reactive methods, as has been done in the beginning. But they are not limited to. Moreover, arbitration mechanisms were developed quickly to tackle the behavior coordination problem. The arbiters have similarities to *hybrid architectures*, featuring a planner or reasoner which utilizes skills or behaviors. There is one strong limiting requirement common for all *BBCs*: as a *BBC* consists of several modules working in parallel, the tasks have to be decomposable. Where “decomposable” is meant in the line of parallel, not sequential, components.

Modern *Behavior-Based Controls* try to address the drawbacks of both, the reactive and functional approaches while incorporating their advantages. The primary strength oft the *BBCs* is their flexibility achieved by modularity: they actually can implement reactive as well as deliberative components. From a system-theoretical point of view one could say that a behavior-based architecture essentially is a meta-architecture, implementing reactive as well as deliberative (sub-)architectures in the individual groups or layers of modules.

An advantage of behavior-based systems over purely reactive systems is that *Behavior-Based Controls* are able to store a world model inherently and distributed over the individual behaviors. Even if this world model is only implicitly available, *BBC* are not limited to systems and applications of low complexity. The modular character of the behavior-based approach simplifies the system’s design and makes the control system more easily extendable – new behaviors can usually be added to the control system without effort to adapt the system to new tasks or situations.

But with the modularity new problems arise, mainly the coordination of the behaviors and fusion of the large number of individual data flows. These problems are analyzed in various works such as done by Scheutz [140] and Althaus [6]. The primary problem identified is that a *BBC* is a self-governing system, with all implications resulting from the absence of a central intelligence. Coordinating all individual behaviors is very difficult, therefore *BBCs* have a tendency to indeterministic overall behaviors and oscillations between individual behaviors. Even more difficult is it to ensure that the control stays always goal-oriented. Another

important drawback is the fact that it is often difficult to identify the root-cause of a problem. Errors, especially oscillations, are forwarded throughout the system and are persistent over several control cycles.

The span of proposals for solving the identified problems ranges from special behavior selection mechanism or arbiters (D.Langer) [102] to letting all behaviors run in parallel without action selection as proposed by Brooks [28] or Steels [154]. More proposed solutions can be found at the end of the section focusing on *BBCs* (2.2.3), after discussing the architectures in general in the following sections.

An extensive and very detailed summary on the fundamental concepts and methods of *BBCs* can be found in a technical report written by P. Pirjanian: [123].

### Classification of *BBCs* and behavior coordination

There are four major characteristics of the behavior coordination mechanism by which a *Behavior-Based Control* can be classified. First of all (1) the coordination can be performed by a central arbiter behavior or it can be done inherently by the structure of the *BBC* itself. (2) the *BBC* can be hierarchical or not. (3) the fusion mechanism can be cooperative or competitive and finally (4) there are several means by which the contribution of the individual behaviors is evaluated and fused, e.g. how important the contribution is for the overall behavior. There are five basic methods: *priority-based*, *activity-based*, *fuzzy-based*, *election-based*, and *state-based*. These four concepts are introduced below:

#### 1 Behavior coordination

**1.a) Central arbiter:** A central arbiter is responsible for the coordination of the individual behaviors.

Popular arbiters are state machines or symbolic planners. The advantage is obvious: the overall behavior of the control is much more predictable and easier to control. Only one single module has to be analyzed to identify the root cause of a problem. The drawback on the other hand is that a central arbiter somehow contests the concept of a *BBC*, reducing an advantage of most *BBCs*: they do not need one central, monolithic and omnipotent algorithm, making them expandable, robust and modular. The question is where to draw the line? There are architectures such as the 3T architecture or the ones used by the robots *Toomas*[66] or *Robovie*[82] (see Section 2.4.3) where a set of “skills” or “behaviors” is defined which are executed by higher level (partially even symbolic or logic) planners. The individual behaviors have no influence if they are activated or not. But, again there are “evaluation” modules which evaluate the contribution of the basic modules. The line is difficult to draw ... a definition is attempted below.

**1.b) Self-governed:** Here no central arbiter is present and the arbitration or coordination is done inherently by the structure of the architecture, often in form of a *Behavior Networks*. The advantage is the high degree of expandability and modularity as individual behaviors can be added or removed without having to adapt a central intelligence module. But apparently the drawback is that the resulting behavior of such networks is very difficult to predict or analyze precisely.

The author of this thesis proposes the definition that a control system is actually a *Behavior-Based Control* when the individual behaviors can try to contribute or to activate themselves autonomously and independently. A central arbiter on the other hand should be able to reject the contribution of a behavior only if other behaviors want to inhibit the former behavior. Meaning that a central arbiter in the line of a channeler for the behaviors' votes can be part of a *BBC*, but in contrast a central arbiter as ruling central intelligence

which independently selects behaviors can not be part of a *BBC*. Architectures featuring a *BBC* for a specific application often need a high level planning instance. These can be called **hybrid architectures**, as long as the central planner is restricted to enabling or disabling whole groups of behaviors, and the coordination inside each individual group is done by the behaviors themselves. Appropriately, the creators of *Robovie* (Ishiguro et al. [82]) distinguish their architecture from the *BBCs* because of this reason.

## 2 Structural organization

- 2.a) Hierarchical:** A *BBC* can be organized in hierarchies, setting up some kind of processing pipeline where the control data is refined stepwise. Often the abstraction level of the required information and the generated commands rises in parallel with the hierarchy of the behavior. Additionally, this concept inherently applies some kind of priority: lower behaviors react faster than higher ones. This approach makes the *BBC* more predictable and easier to plan and to set-up.
- 2.b) Homogeneous:** Here the *BBC* is made up of many modules which are all equally important and are treated in exactly the same way, resulting in a simple control architecture. But the behavior of the overall system is much more difficult to predict.

Often at least one of the two methods for reducing the level of self-organization of the behaviors is found. Or the control system is *hybrid*, featuring a *BBC* for reactive tasks and a classical plan-based approach for higher ones. Pure homogeneous *BBCs* are seldom found, at least in application-oriented systems where tasks have to be fulfilled or interaction with operators is necessary.

## 3 Cooperative vs. competitive

- 3.b) Competitive fusion:** In case of a conflict between behaviors, one single behavior is selected to contribute its output, solely providing the control for the overall system.
- 3.a) Cooperative fusion:** All behaviors contribute to the overall behavior. Obviously, some kind of weighting mechanism is needed to determine which behavior shall have how much influence in the current situation.

It is usually difficult to determine which method is the better one, as for both exist situations in which they excel as well as fail. The design of the individual behaviors as well as the remaining components have to take this into account. Figure 2.6 shows three exemplary situations where the robot shall drive past an obstacle. The arrow indicates the path taken by the robot. The left pair of sketches shows that usually the cooperative methods provides smoother trajectories. Using the competitive method only, one behavior is active at a time, here resulting in the sequence ToTarget, AvoidObstacle, TraverseObstacle, ToTarget without smooth transitions. The middle pair of the sketches shows that in the cooperative case behaviors can negate each other, here the behaviors AvoidObstacleLeft and AvoidObstacleRight, resulting in a collision. The third pair of sketches shows that in a competitive case the robot could take a wrong way when a wrong behavior is too strong even if only for a short period of time. Here the AvoidObstacleLeft overruled ToTarget and AvoidObstacleRight, so the robot moved left.

## 4 Fusion method

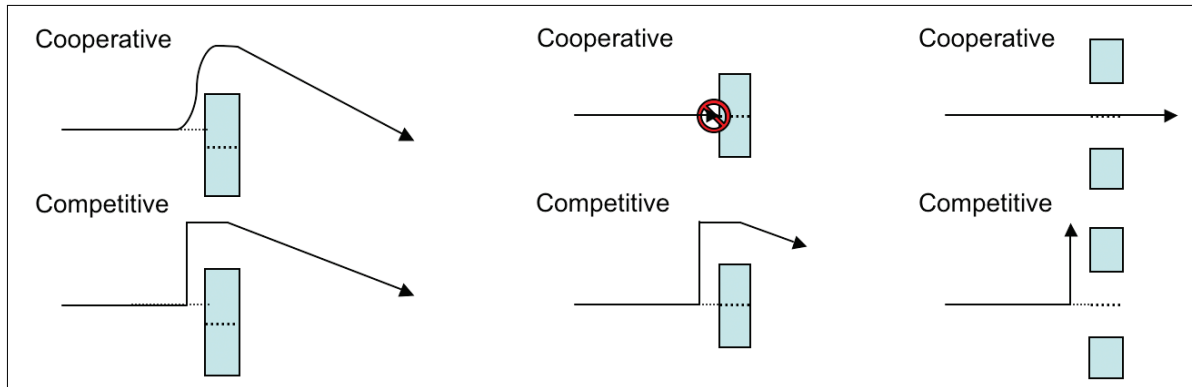


Fig. 2.6.: Fusion mechanism for *BBCs*: Exemplary results for cooperative and competitive *BBCs* at the example of obstacle avoidance. Left: smoothness of path. Middle: behaviors negating each other. Right: behaviors overruling each other.

**4.a) Priority-based:** Each behavior has defined priority. This priority can be fixed or can be allocated dynamically by an arbiter, a state machine or similar.

Cooperative case: The contribution of each behaviors is weighted by its priority

Competitive case: The contribution of the behavior with the highest priority is taken, the remainder is discarded.

**4.b) Activity-based:** Each behavior evaluates the worth of its own contribution or respectively how much it “wants” to contribute.

Cooperative case: The contribution of each behaviors is weighted its activity

Competitive case: The contribution of the behavior with the highest activity is taken, the remainder is discarded.

**4.c) Election-based:** This is a more complex scheme: a limited set of possible actions is defined. Each behavior votes for (or against) individual actions. There are several variants: voting for or against one or many actions. Each behavior can have votes of the same value or a value defined by its activity of depending on a fixed value (priority). Each activity can have an accumulator of votes which is reduced by a fixed value each cycle, resulting in a more stable system, but with bad reaction times.

Cooperative case: All actions are executed, weighted by their (accumulated) votes (this case rarely found)

Competitive case: The action with most (accumulated) votes is executed

**4.d) Fuzzy-based:** This is an alternative approach, skirting the problem of behavior coordination. Coming at the cost of getting the problem of de-fuzzification later on. Each behavior represents a fuzzy rule, contributing a set of (truth) values for the overall function. In the defuzzification step all these values are combined by using either an AND or OR combination of the individual contributions, and finally a “crisp” output value is generated. There are several approaches for the defuzzification. The most popular are: *center of gravity*, *center of largest area*, and *global maximum*.

Cooperative / Center of gravity: The action lying in the center of gravity of the largest connected fuzzy set is taken



Cooperative / Center of largest area: The action lying in the middle of the largest connected fuzzy set is taken

Competitive / Global maximum: The action with the absolute highest value is executed

**4.e) State-based:** A state machine is used for coordination. This method is often used to adapt other methods to the current state of the system by allocating weights or priorities, by (de-) activating groups of behaviors, or even by switching between coordination methods. When this method is the sole method of coordination, the architecture is on the verge of leaving the group of the *BBCs*, see the discussion on the central arbiter above.

In the various existing control architectures these five methods are often mixed amongst each other or with other concepts to tackle various drawbacks or to adapt them to a specific application.

### Popular approaches

After having discussed the characteristics of *Behavior-Based Controls (BBC)* in general, this section will give a brief overview of popular approaches when actually implementing robot control systems. The *Subsumption* architecture by R. Brooks from 1986 [27] is the most well-know approach in behavior-based robotics, as it was the first architecture which enabled programming (relatively) complex robots using simple reactive behaviors. While the behaviors were only capable of performing local reaction to sensed inputs, the architecture offered the mechanisms for coordinating behaviors and aggregating their possibly conflicting responses. A robot can then cope with various configurations of the environment, by having different groups of behaviors activated in different situations. Although the *Subsumption* architecture provided interesting results in making robots more efficient in real-world environments, a major limitation was that managing the interactions and side effects between behaviors becomes too complicated even when few behaviors are involved [57]. Additionally, this architecture is hardly scaleable or expandable – usually an advantage of *BBCs*.

The coordination of the individual behaviors is the crucial point when designing a control system. Accordingly, authors have analyzed different methods to tackle this problem. Scheutz [140] for example distinguished between cooperative / competitive, explicit / implicit, and nonadaptive / adaptive coordination methods. A large number of approaches was implemented over the time: some are using more static arbitration mechanisms like priority-based (e.g. *Subsumption* architecture by Brooks [27]), state-based (Lang [101], Arkin [9]), or winner-takes-all-based (Newell [119]) mechanisms. More dynamic approaches are fusion methods like fuzzy fusion (Saotti [136]), voting (Rosenblatt's DAMN [134]) or superposition based fusion (e.g. potential fields [88]), and nonlinear dynamical systems (Althaus [6]). Rosenblatt presents an approach using a central arbiter with a large amount of knowledge about all behaviors, which coordinates all behaviors, the *distributed architecture for mobile navigation (DAMN)* [134]. Contrary to this approach of a steered coordination, Steels [154] and Brooks [28] proposed control systems completely without coordination mechanism, as it will be taken up by the *Behavior Networks*. The drawback here is that in both approaches from the early 1990th in each behavior detailed knowledge on other behaviors had to be incorporated, resulting in significant overhead and complex behaviors. Saotti's fuzzy-based ([136]) and Jaeger's differential equation-based ([83]) approaches both offer the advantage that a formal analysis of the control systems is possible. But they also limit the types of algorithms which can be implemented in form of behaviors.

J. Bryson ([29]) introduces the *behavior-oriented design (BOD)*: a development process for modular architectures, with the requirement that control- and coordination data flow shall be separated. It proposes an action selection mechanism driven by reactive plans. Defining such design processes is an important step when developing a *BBC* to be able to keep the control predictable. Accordingly, many authors have taken similar approaches.

An approach to separate the control data flow from the coordination data flow was followed by using some kind of signals (activity, target rating, motivation and inhibition values) for each behavior. The approach also introduces simple fusion behaviors to merge different data flows. This approach is called *Behavior Networks* and originates from work by J. Albiez for the behavior-based control of walking machines [4]. Further developed versions are used to control walking machines like LAURON or BISAM [2] or wheel driven robots like RAVON [139]. On LAURON the *BBC* is used for controlling the motion only and higher tasks are planned and executed by classical approaches as shown for example in [180] where a semantic planner for LAURON is described. The architecture used on RAVON consists completely of behaviors, even in the higher task-oriented levels of the control system. As the *BBCs* by J. Albiez and K. Berns are related to this thesis, they are explained in more detail in the next section.

### **Behavior Networks by J. Albiez and the iB2C architecture**

The *Behavior Networks* were developed by J. Albiez [3] here at the *Research Center for Information Technology – Forschungszentrum Informatik (FZI)* in Karlsruhe to control walking robots. They implement a hierarchical *behavior-based* control system which strictly distinguishes between the control data flow for commanding actors and the coordination data flow for coordinating the individual behaviors. The individual behaviors are software modules which are implemented by *modules* of the *Modular Controller Architecture (MCA2)*[143]. Each behavior is dedicated to an individual task. All behaviors of such a control system work independently and parallel at all the time. Besides calculating the dedicated output ( $u$ ) based on given input ( $e$ ), which represents the control data flow, each behavior has the possibility to interact with other behaviors directly, which represents the coordination data flow. A behavior can be activated ( $t$ ) by other behaviors, and each one can inform others about how active ( $a$ ) it is and how satisfied it is with the current situation ( $r$ ) (see Fig. 2.7). This information can be used by other behaviors to estimate how efficiently the first one is working and to motivate or inhibit it accordingly. Using these capabilities, individual behaviors can be woven into *Behavior Networks*. In such a network a whole set of behaviors combine their strictly dedicated abilities to fulfil a higher task. Additionally, there is a special kind of very simple behavior module, the fusion behavior. These unify several inputs of the same type to one output of the same type. This is necessary when several modules provide input for one individual other module (see Fig. 2.8). The most prominent types are the maximum fusion, where one input is selected based on the highest activity of the source modules as well as the weighted fusion where all inputs are weighted by the individual activities and then merged for the output.

**The iB2C architecture** The *integrated Behavior-Based Control (iB2C)* architecture is being developed by K. Berns and his group at the University of Kaiserslautern (Germany)[125]. It is based on the *Behavior Networks* developed by J. Albiez to control walking machines. They enhanced the *Behavior Networks* so that they can also be used for more deliberate tasks. Finally, they even implemented the complete control architecture including high-level planners by behaviors. A wide range of methods were implemented by

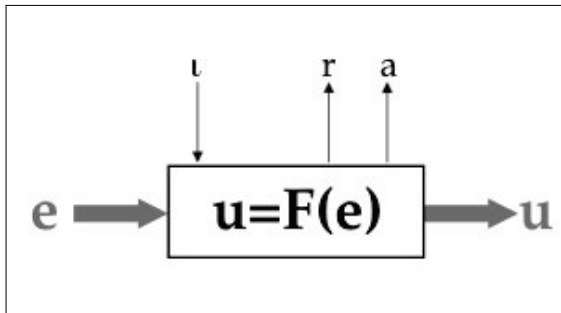


Fig. 2.7.: Behavior Module [3]

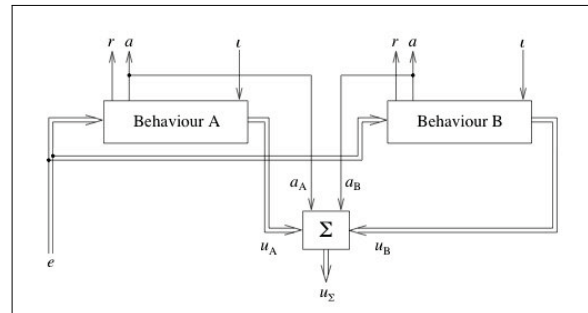


Fig. 2.8.: Linking of behaviors by fusion-behaviors [3]

behaviors: classical behaviors for obstacle avoidance like done for the robot RAVON (see sketch in Fig. 2.9 and [10]) but also complete algorithms like  $A^*$  or elastic bands for the robot ARTOS[18], demonstrating the flexibility of the *Behavior Networks*. (More details on the two robots can be found at the end of Section 2.4.1.)

The *iB2C* architecture inherits several features from Albiez's *Behavior Networks*: the separation of information- and control data flow, the coordination of the behaviors by special fusion behaviors as well as the standardized inter-behavior communication using defined signals such as *activation* ( $a$ ), *inhibition* ( $i$ ) or *motivation* ( $m$ ). Just as the architectures of Albiez, *iB2C* does not use a central world model.

One of the primary concerns of the authors was to develop a *Behavior-Based Control* which behaves in a truly predictable way. Therefore, they defined a set of design principles which shall in combination with the characteristics of the *Behavior Networks* allow making assumptions about the network:

- The activity of behaviors is always smaller than the external activation
- Goal state activity: when  $r$  becomes 0 the activity is maintained (const) because behavior shall try to hold the system in the goal state (prevents oscillation caused by external forces e.g. joint angle and torque of a manipulator vs gravity)
- $r$  is independent from  $s$  or  $i$  and from the activation
- Fusion behaviors do not inject or drain activation or rating from the system
- A behavior can only be stimulated or inhibited by  $a$  or  $r$  of other behaviors (or from defined sources), there are no "hidden" sinks or sources of activity
- Each control signal shall always be accompanied by a corresponding  $a$  and  $r$ . These must not be dropped until the control signal leaves the network.
- No stimulation or inhibition cycles

To reach the goal of a predictable *Behavior Network* the authors have not been content with defining the design principles. They needed methods to analyze the developed networks. Accordingly, they introduce four mechanisms for analyzing the *Behavior Network* online and / or offline:

1. **Graph visualization:** A graph can be extracted, stored using the boost graph library, and visualized using graphviz. A static graph analysis is possible.

**2. Oscillation analysis:** To detect and trace oscillations of activation signals in the network

- Oscillation detection: A ring buffer stores signals, a FFT transforms the data into the frequency domain for the identification of peaks. Then the data is transferred back to time domain to identify the regularity of peaks. This method has too much computational effort to run continuously on all signals, it is only used on-demand.
- Oscillation tracing: Some initial behaviors look for oscillations. If a behavior detects oscillations it asks all successors and predecessors to also look for oscillations. The gathered information is stored in a graph which can be analyzed. As no circles are allowed, the initial behavior can be identified.

**3. Formal verification:** They performed a formal verification on safety critical behaviors. It should guarantee that the tested behaviors follow three specifications regarding the output set-point velocity. The behaviors were modeled using the QUARZ language which can be transformed into a transition system and then be verified with the AVEREST framework [142]. After verification, C-code can be generated which again is embedded into MCA-modules.

**4. Online Tools:** Two tools are used frequently when working with the MCA framework. These are the *MCA-Browser* and the *MCA-GUI*. These allow looking into the inner state of the individual MCA-modules and edged at runtime.

The authors state that their architecture supports all six methods of behaviors coordination seen in other architectures by using *a*, *r*, *i*, *s*, fusion behaviors, and auxiliary behaviors which implement state machines or similar.

Summarizing, the *Behavior Networks* as well as the *iB2C* architecture can be assigned the following attributes (regarding their usual application): inherently they are architectures without central arbiter using a cooperative and activity-based fusion approach. They can be, but do not have to be, implemented as hierarchical and / or hybrid architectures, whereas *iB2C* usually is not of the hybrid type. But due to their very flexible approach they can be used to implement other types of *BBCs* as well, mostly by using auxiliary behaviors.

#### **2.2.4. Hybrid and hierarchical architectures**

The hybrid architecture approach emerged from the inherent limitations of reactive and / or behavior-based architectures and the need to integrate higher-level, deliberative capabilities of functional architectures [57]. As all approaches, the reactive, the behavior-based and the plan-based one, have their strong advantages, it stands to reason to combine them. The once monolithic architecture is broken up into independent layers, and the individual layers are implemented using different control approaches. Usually a layer housing reactive skills is combined with a layer housing a classical planner. Often an additional layer is placed in between to integrate both. Sometimes behavior-based systems are used in the lower layer(s), but often there are “just” skills which are sequentially utilized by the planner.

The most common realization of the hybrid approach use three layers. Three-layer architectures were introduced with layers for reactive control, for reactive plan execution, and for deliberative computations (Alami [1]), and with a functional level, an execution control level and a decision level (Gat [57]). Three

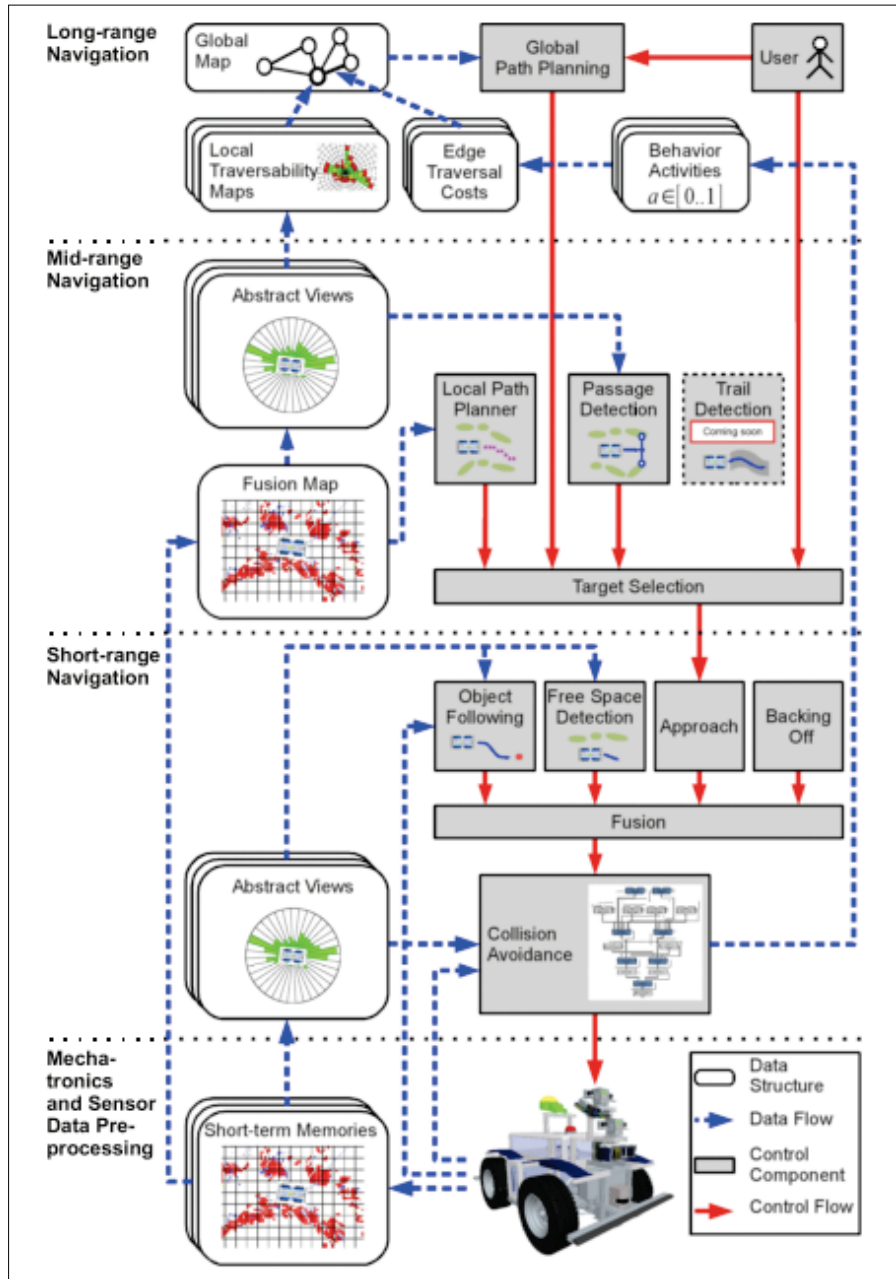


Fig. 2.9.: An exemplary implementation of *ib2c* from 2011 on the out-door robot *RAVON* [10].

layers are also proposed by Kim ([92]). Those layers are composed of deliberate, sequencing, and reactive layers based on the hybrid approach. The purposes of the deliberate layer are to interface with a user and to execute a planning process. The sequencing layer is classified into two groups, that is, the controlling part that executes the process by managing the components in the reactive layer and the information part that extracts highly advanced information from sensor data. The reactive layer controls the real-time command and hardware-related modules for sensors and actuators. Kim also presented a robot architecture with the intention for the rapid development of robot systems with different requirements with emphasis on reusability and extensibility [90]. Their architecture is a hybrid architecture consisting of three layers: reactive layer for real-time services, sequencing layer which supports primitive behavior, and deliberative layer for task managing. While a lot of recent robotic architectures are inspired by the hybrid architecture approach, the necessity of the three classical layers was questioned. For instance, some robots were built according to a two-layer architecture (Nenas [118]) with a functional and a decision layer (only). In this approach, the decision layer integrates decision and plan execution more tightly to take into account the coupling between their tasks. The next section exemplarily analyzes two well known instance of three-layered architectures which have been used as an archetype for many robot control systems: the *3-Tier* architecture, combining reactive skills and a reasoner and the architecture proposed by R. Alami.

### The 3T architecture

This hybrid architecture, incorporating a planner and reactive skills, shall be picked up here to be explained in more detail, as it has been a source of inspiration for many hybrid and hierarchical architectures, including the one presented in this thesis.

The *3T Architecture* [22] – where “T” stands for “tier” – is an hierarchical control architecture which incorporates a classical reasoner in the upper tier and reactive skills in the lower tier (a sketch can be found in Fig. 2.10). The *sequencing Tier* concatenates these two by breaking the plans produced by the reasoner down into atomic actions and then activating skills accordingly. The environmental situation is observed by dedicated monitors to identify the fulfillment of clauses representing certain conditions relevant for the plans. The three tiers will be briefly introduced:

1. **The *deliberative Tier*:** This tier houses several action-sets called *RAPs*. Each of these is linked with certain clauses for pre-conditions and effects. A reasoner generates a plan based on the available *RAPs*, their clauses as well as the clauses describing the current and the target environmental situation.
2. **The *sequencing Tier*:** This tier decomposes the plan in sub-plans and actions and executes them sequentially. Execution means in detail that corresponding skills and monitors in the *reactive Tier* are activated. Once the monitors communicate that the fulfilment-clause of the action is true the tier proceeds with the next action. Exceptions are handled in a similar way: for every exception there is a corresponding monitor observing the situation.
3. **The *reactive Tier*:** This tier houses the skills and the monitors. Skills are software modules which either process sensor information or generate motor commands. They represent the interface to the specific robotic platform and have a defined command- and data-interface to ease the integration with the *RAPs*.

The *3T Architecture* itself is platform- and application-independent. But the major components which are implementing the architecture are not: the *RAPs* are application-specific and the skills are platform-specific. The architecture has - sometimes only partially - been implemented in several robots.

### “An Architecture for Autonomy” by R.Alami

Another well known hybrid architecture known as “An Architecture for Autonomy” was proposed in 1998 by R. Alami [1] along with a corresponding design approach. Most remarkably, the major characteristic of this architecture is that the functionalities are modeled using description languages and the actual code is then generated automatically. The architecture uses overall five levels while having the popular three-leveled setup of the central functional part: operator, decision, execution control, functional, and logical system level (see sketch in Fig. 2.11). The *decision level* houses the intelligence of the system, the lower levels are used for task refinement and act as library of services.

1. **The *Operator Level*:** This layer provides the user interface and maps the queries to the available missions.
2. **The *Decision Level*:** this level contains an supervisor which controls task execution and utilizes planners. The supervisor tracks the state of the world and provides this information for the planner. The planner then generates a task-graph based on atomic plans chosen from a database corresponding to actual or estimated world states. The used planner is the temporal constraint solver “IxTeT” which resolves resource conflicts – the robot’s function are interpreted as the resources.
3. **The *Execution Control Level*:** this level actually manages the translation between the levels two and four: it fills the gap between symbolic plans and numeric computations while itself being purely reactive, it does not perform any planning itself. It receives the plans, actuates the functions based on a database for mapping functions and action and finally reports to the decision level. The level is implemented by an automaton which is generated automatically by “GenoM” descriptions and the description language “Kheops”.
4. **The *Functional Level*:** This level embeds the libraries of functions which are implemented by modules. These are generated by the “generator of modules” – “GenoM”: it is used for formalizing the instantiation of modules to reduce implementation errors using the dedicated description language “Kheops”. The modules are based on a generic module, consisting of two functions: controller (actually a state machine) and the execution engine as well as of two databases for parameters and state of activities. Each individual module is activated on request by the plan, but once activated they work independently to increase robustness. They communicate using defined areas of shared memory.
5. **The *Logical System*:** This is essentially the hardware abstraction, containing interfaces for sensors and effectors which are modeled into the modules.

### 2.3. Navigation, self-localisation and mapping

After having had a look at control architectures we now will have a closer look at the methods which provide the functionalities inside the control architecture. This section summarizes work related to the individual

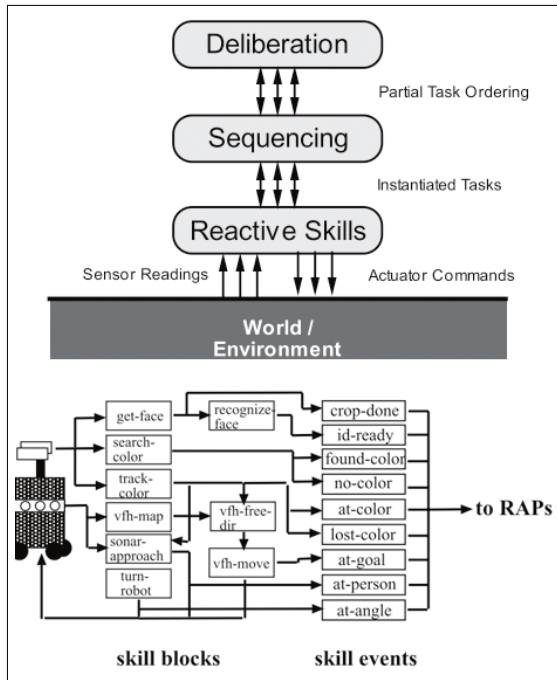


Fig. 2.10.: The three-layered *3T Architecture* containing a symbolic planner and reactive skills [22]

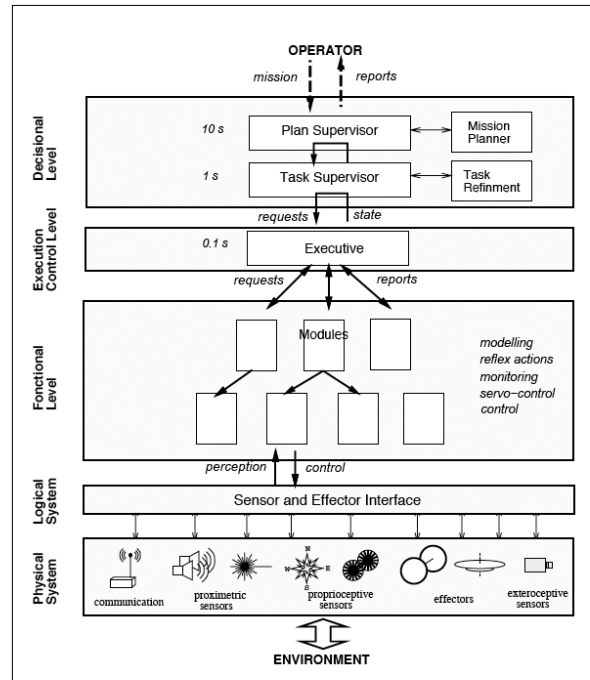


Fig. 2.11.: The architecture proposed by R. Alami with five layers containing symbolic planners and reactive functions [1]

components utilized or developed in this thesis which correspond to the navigation system. It starts with a summary on RFID-based self-localization, continues with mapping which again is followed by the avoidance of static and moving obstacles. The section finally finishes with the topic of path planning methods. Because of the number of topics the individual ones can only be briefly introduced as far as necessary for the understanding of subsequent chapters, much more details can be found in standard literature as for example in *Probabilistic Robotics* by S. Thrun, W. Burgard, and D. Fox ([164]).

### 2.3.1. RFID-based self-localization

RFID technology is very attractive when tackling robotic problems like object recognition, topological localization, or person tracking. Many problems can be made simpler assuming that RFID transponders are incorporated into objects, are embedded on mobile agents (persons, robots, vehicles) or are distributed in the environment. The tags can be detected robustly and they provide a unique ID along with the data which is stored on them. Many robotics researchers tried to take advantage of this technology using either active RFID tags (long range detection) or passive ones (short range) utilizing either omni-directional or directional antenna systems. The drawback is that the tags can not be located precisely – it is known only which antenna detected them and maybe the distance, but not the angular position. Additionally, the tags can be occluded by metallic or water-containing bodies, therefore by many devices as well as by human beings.

To tackle the drawback of poor localizability of the tags, a popular method is to place the tags on the ground with a very short – and this way defined and invariant – distance between antenna and tag. Because the distance is known, and the size and shape of the antennas FOV at this distance is known too, the position of the tags can be estimated, especially when several tags of a densely tagged environment are seen at the same time. Even more, by using small-sized antennas with a very narrow field of view, the tags can only be



detected when the antenna's center is exactly above the tags. Re-writeable lines of RFID tags were proposed by Bosien [26] as substitute for optical lines or electric wires integrated in the ground that the robot could easily follow. The lines could be modified by changing the data stored on the tags to gain flexibility. This approach has been generalized with RFID-carpet on which the robot is always able to localize at least roughly. Several authors have studied how to use classical probabilistic methods for self-localization in combination with RF-technologies like S.Thrun [159] using EKF-based methods. Markovian- and Monte-Carlo-based methods were proposed to estimate the robot's position like done by Haehnel ([68]), Vorst ([175], [176]), or Mehmood ([110]).

An approach for densely tagged environments is presented by P. Vorst [174]. Often it is assumed that RFID-based self-localization is too inaccurate, but P. Vorst has shown that using new methods based on particle filtering [174] that an accuracy of less than 0.3m is possible. M. Devy has shown that precise self-localization of mobile robots is possible using RFID-tags which are sparsely distributed in the environment and a ring-shaped setup of long-range antennas ([128], [37]). In densely populated areas both methods run the risk, that the robot is completely screened by humans and therefore no tags can be detected.

The sensor modeling step has an important impact on these methods. Vorst ([176]) developed simplified procedures to estimate the antenna model based on previous collected data; they create a cartesian grid of relative position-detection probability. In addition a more complex method considering the measurement density is presented, taking into account correlations between the probabilities to acquire measurements in every block. This approach is interesting because it is difficult to get a data set with all blocks having a satisfactory number of measurements.

### 2.3.2. Maps and mapping

The automatic generation of environmental maps is a very wide field challenging many groups of researchers. As mapping is not part of this thesis this topic will be omitted here mostly. A widespread description was summarized for example by S. Thrun in the work "Robotic Mapping: A Survey" [161].

Usually three major types of maps are used: metrical, topological, and semantic maps, with rising level of abstraction. *Metrical maps* often consist of occupancy maps but can also use geometrical representations of obstacles or free space. *topological maps* represent the topological layout of an area, divided into subareas without containing information on the sub-areas themselves. To enhance the applicability of *topological maps* they are often used in hybrid versions (see Fig. 2.12): The *topologic-metrical map* contains rough metrical information on the sub-areas such as size, relative positions, or hotspots. The *topological map with local metrical maps* is a hierarchical structure where local *metrical maps* are stored for each topological sub-area. *Sematic maps* contain abstract information. They are often used to annotate *metric maps* or *topological maps* with semantic information, e.g. stating that a specific topological area is a "living room". The semantic information can then be used to trigger behaviors or to communicate with humans.

#### Metrical mapping

Usually the occupancy grid / certainty grid method are used for local map building [31]. There are numerous ways of calculating the occupancy probability of a cell. The range is from the probabilistic modeling of the sensor systems [160] to the learning the probabilistic distribution with neuronal networks demonstrated by Burgard [31]. An alternative approach is to represent the geometry of obstacles as polyhedra [34] or to

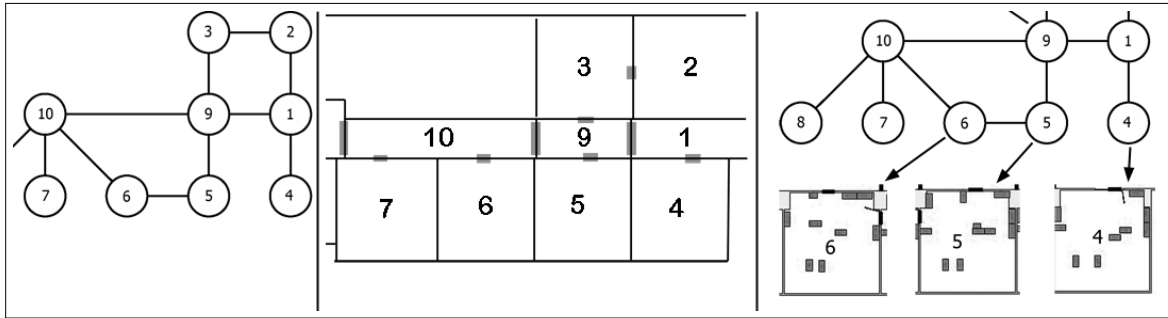


Fig. 2.12.: Three variations of *topological maps*. Left: the basic *topological map*. Middle: the term *topologic-metrical map* is used here for *topological maps* containing metrical information such as sizes or relative positions of the areas but no detailed or dense data (these would be called “patchwork metric map”). Right: *topological map with local metrical maps*

construct objects by extracting wall segment from raw sensory data [158]. S. Thrun has shown in [165] a statistical method that revises the robot's pose forward and backwards in time. It was developed to build maps of large areas where it is difficult to estimate the exact pose of the robot.

Recent research concentrates on 2D and 3D *SLAM* of various mobile agents in dynamic environments, including approaches for *life long mapping*. Additionally, the mapping process is used to extract topological or even semantic information as for example described by J. Oberlaender ([120]).

## Topological mapping

Topological maps are very compact because they contain sparse information only, compared to metrical maps. They represent large structures like corridors or rooms so they are (almost) time-invariant as shown by P. Althaus ([5]). This makes them a very good choice for highly dynamic environments, where even parts of the structures are altered (compared to environments with moving objects where the structures are static). Additionally, they are easy to construct even for large places. When considering human robot interaction, the facts stand out that *topological maps* are much easier understood by humans than large metrical ones. A human asking for directions will expect something in the line of “go straight on and turn left at the third corridor” instead of “go 11.34 m ahead and then turn -90 degree at the pointed feature”. Often *topological maps* are combined with local metrical maps as described by B. Kuipers ([100]). Sometimes *topological maps* are constructed manually where they could be used to support an exploration process [116], but often the robot learns them automatically [159] while exploring autonomously. J. Oberlaender ([120]) describes a mapping process where topological and semantic information is extracted, which is even able to handle topological or semantic features which are significantly larger than the robot's sensor range. Based on semantic maps, semantic navigation can be performed as described by K. Uhl in [169]. The advantage here again is the robustness against environmental changes and the even more easy comprehensibility by humans.

### 2.3.3. Collision avoidance

This section summarizes current and past works on the topic of collision avoidance. It starts with static obstacles and continues in the succeeding section scenarios with obstacles which are moving themselves. For several methods it is hard to distinguish whether they fit into the section “collision avoidance” or “path-

finding and navigation” as they combine direct reactions to obstacles and goal driven characteristics. In these cases the methods shall be at least named in both sections.

### Collision avoidance with static obstacles

A classical approach to collision avoidance, or navigation in general, are the class of potential field methods where obstacles have a high potential and flat terrain and especially the goal has a low potential. The robot then plans a path following the steepest gradient. Introduced by B. Krogh in 1984 ([98]) and taken up by O. Khatib in 1986 to generate real-time obstacle avoidance control ([88]), these have become very popular as they are easy to design and fast to implement. Due to limitation of the range of the field of view, potential field methods are computationally highly efficient. But they have limitations like trapping situations or oscillations because of the purely reactive approach, the restrictions in the range of the FOV, and the inability to use backtracking or similar approaches (e.g. Koren [96]). Hence, these methods are often enhanced like done by O. Khatib in [89].

An alternative approach, which also operates directly based on the occupancy map, is the *Virtual Force Field (VFF)*, introduced by Borenstein ([24]). Some sketches of this and the succeeding methods can be found in Figure 2.13. In the *VFF* approach, each cell in the “active area” of the grid exerts an repulsive force vector on the robots with a force proportional to the distance to, and the occupancy certainty of, each individual cell. The target exerts an attracting force vector. All these numerous vectors are summed up to provide the resulting set-point vector for the robot’s drive system.

Due to drawbacks resulting from the harsh information reduction and the discretization of the robot’s position, the *VFF* were developed further to the *Vector Field Histograms (VFH)* ([25], [170]). Here the active area is divided into several narrow sectors defined in polar coordinates. For each sector the weighted sum of the individual cells’ distance to the robot and the certainty of their occupancy is calculated. This way, a 2D histogram is generated, containing for each discrete span of driving directions a value representing the degree of danger of this direction. The “valleys” in this histogram, with a bottom below a defined threshold, are the options for the robot to choose from. The robot selects the valley fitting best to the direction of the target. In narrow valleys the robot chooses exactly the middle line, in wide valleys the robot can choose between each of the two borders and the direct way to the target, if applicable.

This approach was further improved, resulting in the *VFH+* [170]. Here the size of the robot is explicitly taken into account by enlarging the obstacles. This is not done in the occupancy map but in the 2D histogram to reduce the computational effort. An hysteresis is applied to prevent the robot from oscillating and finally the histogram is masked to block valleys whose direction the robot cannot turn into due to its current velocity. The methods from *VFF* to *VFH+* are all purely local algorithms, resulting in not optimal paths due to a lack of knowledge and planning.

By developing the *VFH\** [171] the *VFH+* is extended with a forward planning step: for each possible direction (valley) of the *VFH+*, a defined time-step is taken, virtually moving the robot forward. Then the *VFH+* step is performed again. This search is performed corresponding the parameters search depths, branching factor, and step width. Obviously, this algorithm depends on an occupancy map which is significantly larger than the robot’s active area. Depending on the search depth and the known area of the map, this algorithm can be used for local collision avoidance or for global path planning.

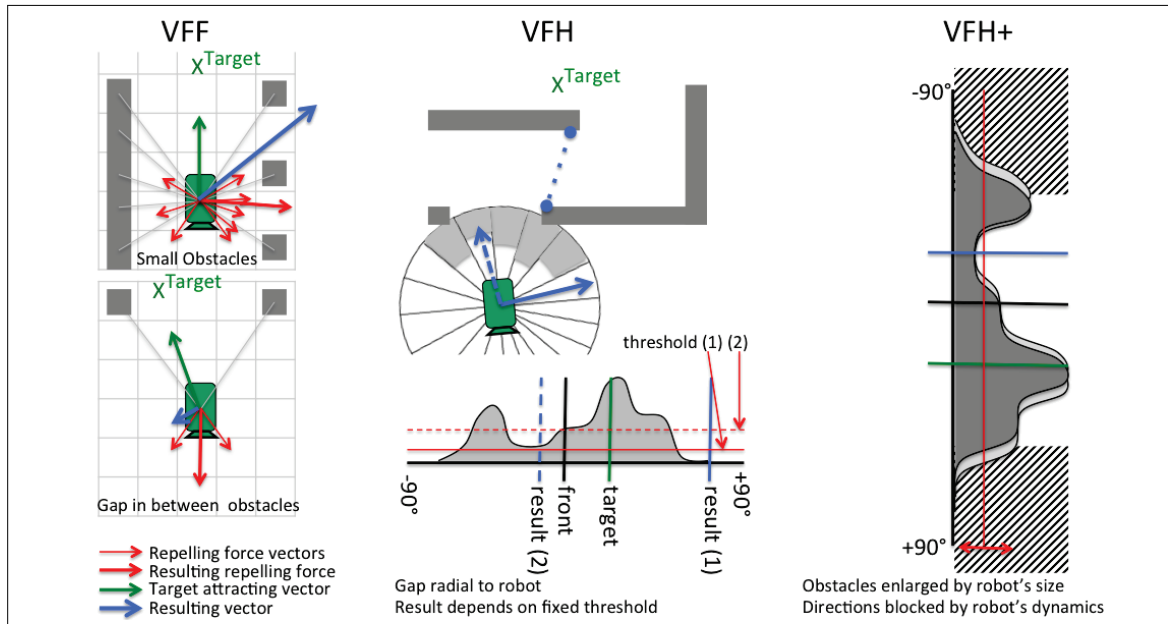


Fig. 2.13.: The function of VFF, VFH, and VFH+ illustrated by drawbacks.

A very intuitive method, which combines obstacle avoidance and path planning, are the visibility(-graph)-based methods. Here the robot plans a path on a known (or learned) map via sub-goals on the visible corners of obstacles, thus avoiding the obstacles (a comparison between different implementations can be found in [105]). This method can be applied to a local map for collision avoidance or on a global map for path planning.

The *Dynamic Windows* method ([47], [46]) for collision avoidance is special compared to the other methods as it directly works in the velocity space. It was designed to cope with higher velocities where the robot's dynamics must not be omitted and where the robot shall always drive as fast as possible. The method generates a search space of possible 2D velocities for the robot. Velocities are considered only if the robot can reach them in a short time window – the “dynamic window”. Areas of the search space are blocked if driving this velocity would result in a collision with objects. Therefore the robot's path as well as the estimated paths of moving obstacles are predicted. Finally a 2D set-point velocity for the robot is chosen corresponding to three criteria: maximum value of the velocity, direction towards the target, and safe from collision.

A completely different approach are the *Elastic Bands* ([127], [137]). These optimize a given path acquired by other methods – often visibility graph or  $A^*$  methods – according to static or even dynamic obstacles. Bubbles are placed on the given path – often with defined distances like pearls on a string, but there are numerous other methods of distributing the bubbles. Then the bubbles are grown until they intersect with an obstacle. When hitting an obstacle the bubble is forced aside while growing further, altering the center point of the bubble. This procedure is performed until a bubble intersects with obstacles on opposing sides or reaches a defined maximum size. The new line of center points of the grown bubbles gives the new optimized path for the robot.

## Avoiding collisions with moving obstacles

After describing methods for the avoidance of static obstacles, the methods for avoiding obstacles which are moving themselves are the next logical step. In the literature the term “dynamic obstacle” is often found – but this term is not unambiguously: it is used to describe objects which can be moved (in the sense of the reliability of a map) as well as for objects which currently move themselves. At first glance these two seem very similar, but they have different implications: in the first case only the navigation and path finding is affected because obstacles can appear which are not mapped, in the second case even a robot can be involved in a collision which is actually standing still, requiring for proactive actions. In this thesis the term “dynamic obstacle” will be omitted in favor of the term “moving object”. “Object” is preferred over “obstacles” as the “moving object” can also be a person or even the robot’s operator which should not be devaluated to be just an “obstacle”.

There are several different popular methods for avoiding moving objects so far. They can be grouped in two main classes: the local or reactive approaches which are limited in their ability to solve complex situations and the global or plan-based approaches which need more knowledge about the situation and where the reaction time can become an issue.

**1. The reactive approaches:** An approach using a multisensor based environment prediction is described by Song [151] where the predicted obstacle positions are used to calculate virtual forces to decelerate the robot. Castro ([32]) presents a system which uses a laser range finder-based obstacle detection and tracking and feeds this information in an extended *Dynamic Window* algorithm.

**2. The plan-based approaches:** Hu first plans a path considering only static obstacles, the dynamic obstacles are considered only by controlling the robot’s velocity when driving along the planned trajectory [76]. Fraichard [48] and Fujimura [51] add the time dimension to the search-space. However, the major drawback of these solutions is that they assume a complete and deterministic knowledge of the environment. In practical applications they are usually combined with reactive methods in order to avoid unexpected obstacles as for example done by Stachniss [153]. Another very popular approach are *Elastic Bands*. First a path considering the static obstacles is planned and later deformed with the elastic-bands-method to direct the robot around the moving obstacles (Quinlan [127]). Hoeller uses a modified probabilistic path planner to avoid predicted trajectories of a human. These predicted positions block the probabilistic planner from adding a new waypoint near the estimated spatio-temporal positions of the human [71]. Large describes a realtime dynamic obstacle avoidance system learning typical trajectories of moving obstacles and feeding them into an iterative motion planner based on *Velocity Obstacles* [103]. Bennewitz uses learned motion patterns of persons: *Hidden Markov Models (HMM)* are derived to estimate future movements of detected objects. The probabilistic belief is incorporated into the path planning process [14].

Finally the problem can be tackled by planning directly in the velocity space, as [122] and [43] have demonstrated. They model the dynamic environment into a map in the velocity space. As the *Velocity Obstacle* approach relies on a perfect knowledge of the world, Fulgenzi ([55]) proposes a solution called *Discrete Probabilistic Velocity Obstacle (PVO)* that combines the *Linear Velocity Obstacle (LVO)* and the *Bayesian Occupancy Filter (BOF)*.

### 2.3.4. Path planning

There are vast numbers of methods used for path planning. As path planning in the thesis is performed only on a topological level in combination with local methods for obstacle avoidance, in this section only the most prominent ones will be introduced. For an complete overview the standard literature is a more resourceful choice. Several  $A^*$ -based methods like dynamic programming (Howard [75], Bellmann [12]) that minimize some kind of cost for every crossed cell like the occupancy value are found in literature. There is a dynamic extension to the  $A^*$  as well, the  $D^*$  [156]. It was developed to perform fast re-planning if the environment suddenly changes without having to re-plan the complete tree. Another application for  $A^*$ -based methods are graph-based navigation concepts like the robot Robox uses. Here a basic path is computed based on a web of nodes and edges that is placed in the free room of the environment.

Other methods are based on voronoi diagrams ([35],[36]) or on visibility graphs ([105], [84]). Voronoi methods lead the robot through the free room in a central manner where visibility graph methods let the robot move as close as possible to the obstacles to minimize the length of the path. Often a reduced visibility graph is used due to high computational effort [124]. Another possibility to reduce the effort are dynamic visibility graphs that look at a local or active region only [77].

Methods based on *Partially Observable Markov Decision Processes (POMDP)* ([104], [121], [152]) were developed. These take the uncertainty of actions, the robots state and observations into account. Due to the very high computational complexity these methods are bound to problems with small action- and state-spaces. Elastic band methods do not actually perform path planning but path optimization ([127], [137]). They rely on the path calculated by the methods mentioned above. The given path is modified according to obstacles in or besides the path, including obstacles that are new and therefore not mapped but seen by the robot's sensors.

## 2.4. Service robots

The concept of *service robots* is – according the Fraunhofers definition (see Chapter 1.1) – focused on performing tasks for the people. The main characteristic to distinguish these robots from industrial robots is that the latter ones are built for producing goods. But when talking about *service robots*, we again find a wide range of types of robots – in Figure 2.14) a more subtle classification is tried. First of all *service robots* can be grouped in two major groups: The first group is here called *functional service robots*. These are built to fulfill a dedicated function. According to Fraunhofer's definition this function is not related to producing goods. But the robots are performing their tasks independently from humans which contrasts what one might assume when hearing the term *service robots*: robots which are working for and in cooperation with humans. This is the second major group identified here: the *user-oriented* or *social service robots*. For each of the two groups some characteristic examples out of the wide range of systems will be presented in one of the following sections. As they are of special relevance for this thesis, a subset of the *social service robots* will highlighted in a dedicated section: the shopping assistants.

### 2.4.1. Functional service robots

The *service robots* which are here called “functional” have in common that they are *service robots* because their services are not dedicated to producing goods. But still their service is not directly assisting a human. The robot is performing a task instead of a human, hence it is fulfilling a dedicate function.

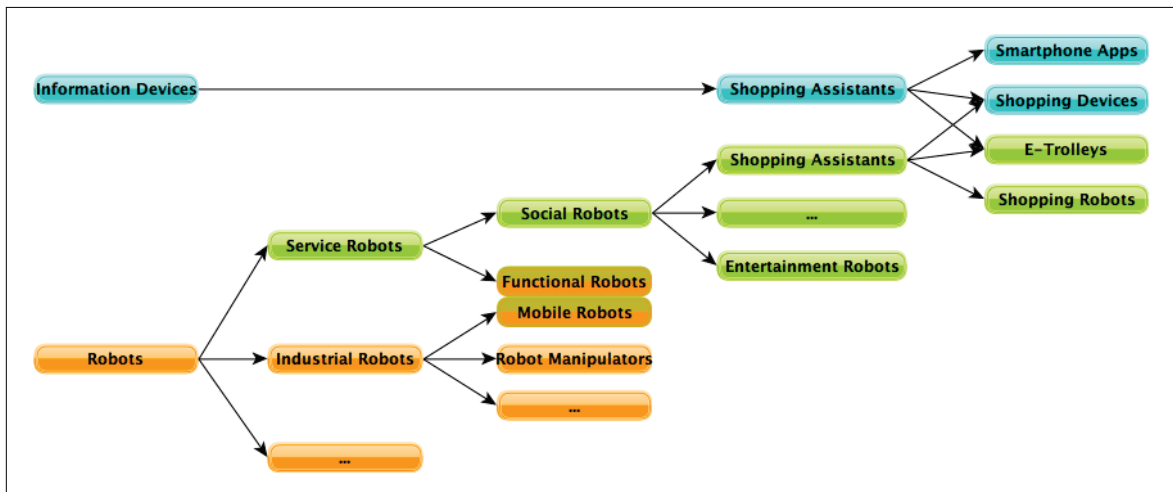


Fig. 2.14.: Hierarchical arrangement of assistance devices and robot types

Many of these robots are cleaning robots like the well known *iRobot roomba* (Fig. 2.17) or the window cleaning robot *RACoon* (see Fig. 1.5) which are rather small compared to the giant *Skywash* (Fig. 2.15) – a robot for cleaning airplanes. Other examples are robots used for inspections like the robot *MOSRO* (Fig. 2.16) developed by *robowatch* which is equipped with gas detection sensors. Yet another example are transportation robots like the *TransCar* produced by the company *SWISSLOG* in cooperation with our laboratory at FZI Karlsruhe (see Fig. 1.3). A special case here is the automatic subway train deployed by *Siemens* in the city of Nürnberg (see Fig. 1.4) because here the service is provided directly for humans. But just like in the other mentioned cases there is no interaction with the humans – they are not recognized as users but either as obstacles or as objects needed for task execution. The robots are commandeered in different ways ranging from central logistic stations in the case of *TransCar* to just giving commands through a control PC. Some can be used for tele-operation. They often drive on pre-defined paths, either virtual ones like used by *TransCar* or marked ones like following color lines, metal lines or magnets. Others again follow patterns – this is obviously used by cleaning robots. In most cases all their systems including locomotion, localization and communication are tailor-made for their specific task and environment.

Fig. 2.15.: Airplane cleaning robot *Skywash* by *Putzmeister* (image source: [126])Fig. 2.16.: *MOSRO* inspection robot with gas detector (image source: [133]) developed by *robowatch*Fig. 2.17.: Cleaning robot *roomba* (image source: [81])

**Service robots using a BBC** Usually *service robots* do not utilize *Behavior-Based Controls*, especially not in commercial applications, due to the high challenges they pose to the developer in guaranteeing their predictability and robustness. But there are some exceptions which are developed at the University of Kaiserslautern (Germany) by K. Berns and his team. As the recurrent term “autonomous” in their names suggests, these robots fall in the category of *functional service robots* as they are used mainly for autonomous transportation, observation or exploration tasks. They all utilize a *Behavior-Based Control*, the iB2C Architecture described earlier in this chapter in Section 2.2.2.

- ARTOS: The *Autonomous Robot for Transport and Service* is under development since 2006 ([109], Fig. 2.18 and webpage: [15]). It is used for the development of methods to integrate robots in *Ambient Assisted Living (AAL)* environments. Possible applications are emergency detection, transportation tasks or telepresence of nursing staff for which the robot is equipped with a camera, a microphone and speakers. For the obstacle detection the robot uses a series of ultrasonic sensors, a laser scanner, and the camera. Laser-based mapping, the popular  $A^*$  algorithm, and elastic bands are used for navigation purposes. All components are implemented as behaviors in the iB2C architecture using the MCA-KL framework.
- RAVON: The *Robust Autonomous Vehicle for Off-road Navigation* is under development since 2005 ([10], Fig. 2.19 and webpage: [17]). Its main purpose is the evaluation of *Behavior-Based Controls* in rough, uneven, and vegetated terrain. Applications could be disaster areas but also patrolling borders or industrial complexes. The main tasks of the system are exploration and mapping as well as driving along routes defined by waypoints. The robot is equipped with horizontal and vertical laser scanners as well as a camera system for obstacle detection. The navigation system is implemented by hierarchical layers using the iB2C architecture: upper layers generate drive commands i.e. driving towards a goal or preferring areas of free space. Medium layer behaviors overwrite these commands and adapt them to the current situation. In the lowest layers there are safety behaviors which can trigger emergency stops based on different events.
- MARVIN: The *Mobile Autonomous Robotic Vehicle for Indoor Navigation* is under development since 2006 ([141], Fig. 2.20 and webpage: [16]). It is developed for the purpose of being a test and evaluation platform for autonomous transport, observation, and entertainment tasks in home and office scenarios. A camera system is used to observe the environment and to detect items of special interest. The operator can control the robot using a touch screen interface as well as via WiFi. The *BBC*, implemented in the iB2C architecture, enables the robot to navigate safely and to adapt to environmental changes. It is able to explore and map its surroundings autonomously. For this purpose a network of individual behaviors like, but not limited to, “Door Driving”, “Narrow Driving”, “Hold Distance”, or “Random Cruise” arbitrated using inhibition and activation signals and is finally merged by maximum fusion behaviors.

### 2.4.2. Social service robots

These are the “true” *service robots* where the term “service” has been taken literally. These robots aim at directly assisting humans. These can be for example elderly people which are the target group of *Care-O-Bot* or customers in hardware stores as in the case of *Toomas*. This section summarizes the most prominent past





Fig. 2.18.: The autonomous transport robot ARTOS (image source: [109])

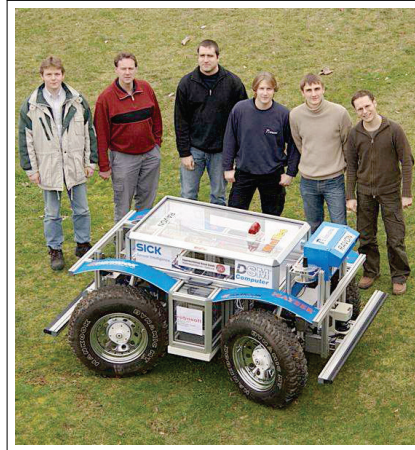


Fig. 2.19.: The autonomous off-road exploration robot RAVON (image source: [17])



Fig. 2.20.: The autonomous inspection robot MARVIN (image source: [16])

and present robot systems which are designed to assist their users. These systems have been used in public environments at least for short term tests, while some have been operating several month. A special section, following the present one, is dedicated to shopping robots as these are designed for the same scenario as this thesis aims on.

### Rhino and Minerva

Rhino (Fig. 2.21, [31], [162]), a robot developed in cooperation by the University of Bonn, the Aachen University of Technology and the Carnegie Mellon University, was deployed for several weeks as museum guide in the museum “Deutsches Museum” in Bonn in the year of 1997. The tasks involved approaching people, interacting with them by replaying pre-recorded messages and displaying texts and images on on-board displays. The goal of the experiment was to gather experiences in the field of navigation in crowded dynamic environments. During daytime *Rhino* had a maximum velocity of 70cm/sec. It was equipped with a laser range finder, sonar sensors and cameras. It did not require modifications in the environment to navigate savely.

*Rhino* used two planners, a motion planner for moving from one exhibit to another, and a mission planner for scheduling tours. The *Rhino* system uses GOLOG/GOLEX for mission planning and execution, which essentially executes pre-programmed plans. The motion planner is a modified version of dynamic programming: each cell of an occupancy grid map has a cost value according to its occupancy probability. Local motion generation and collision avoidance is performed based on a dynamic window approach. The map of the museum contained objects that are invisible to the robots sensor system and had to be fed in by a set of virtual sensors to avoid collisions with the invisible obstacles. The main problem was the reliance on an accurate map. According to the authors, the construction of the map took about a week. But a new algorithm reduced the time by one order of magnitude.

The robot *Minerva* (Fig. 2.21, [162]) was developed based on the robot Rhino an deployed 1998 as museum guide in the “National Museum of American History” for some days. The original robot was especially enhanced to improve the abilities of human-robot interaction and telepresence.

### Robox

The robot *Robox* (Fig. 2.22, [149]) was developed as an interactive tour guide at the Expo02. The challenge to cope with was the robust and reliable navigation in highly populated areas without modifying the environment. During the Expo a whole fleet of these robots was online for 5 month, 10 hours a day. Therefore an crucial design criterion was to depend on a very low level of manual supervision and maintenance.

*Robox* does not navigate based on free-space models like occupancy maps. Instead, a weighted graph consisting of landmarks and geometric primitives is used. Therefore the environmental representation is very compact. The resulting graph contains *station nodes*, *via nodes* and *ghost nodes*. The *ghost nodes* are used as virtual barriers near invisible obstacles. The resulting graph for the Expo consists of 17 *station nodes* and 44 *via nodes*. The self-localization is performed based on a global extension of an *Extended Kalman Filter (EKF)*. The navigation is divided into 3 layers. The first one plans the global path on the graph, the second one plans a local path based on the *NF1 navigation function* which implements a gradient method. Because it tends to generate un-smooth trajectories and graces obstacles, the generated trajectories are optimized using elastic bands. The third layer manages the velocity control and is based on the *Dynamic Window* approach for collision avoidance. The robot was able to escape U-shaped obstacles and to traverse through narrow gaps, supported by Robox's octagonal shape. Especially the division of the "obstacle avoidance into a purely reactive part with high model fidelity and a planning part with local scope" is mentioned by the authors as a "powerful conjunction".

### Care-O-Bot

The Fraunhofer Institute branch IPA develops the service robot family Care-O-Bot since 1998 (Fig. 2.23, [49]). The latest robot, Care-O-Bot 3, is designed to assist elderly people in their everyday environments. Therefore, it is equipped with an robotic manipulator arm and it is able to provide walking aid just like *rollators* (also known as *walkers* or *zimmers*) do. The robot can lead the user to a designated location or it can be steered by the user directly while avoiding obstacles and providing the walking aid. The current version uses an omni-directional drive with four steerable wheels. Using its flexible navigation, the robot is able to move safe and reliably in public areas. It can manipulate and grasp objects using a 7 DOF hand with three fingers.

Communication with the robot is done via a multimodal user interface that includes a touch screen and speech. A hybrid control architecture with a variety of reactive and deliberative components was developed to provide the desired functionalities. These are gathered in the *robotics toolbox*. The highest layer houses a symbolic planer; self-localization is performed by a combination of odometry and landmark-analysis. The path planning can be performed with one out of a set of methods that include *Rapidly-exploring Random Trees (RRT)*, potential fields and visibility graphs. Path planning takes place based on a metrical exact map of the environment that can be acquired by *SLAM*. Avoidance of obstacles that are not mapped and path optimization is done by *elastic bands*.

**Fujitsu Enon** The robot system *exciting nova on network – enon* (Fig. 2.24, [52]) is developed and sold since 2005 by the two companies Fujitsu Frontech and Fujitsu Laboratories. It is a multi purpose *service robot* designed to fulfill several tasks in the peoples' daily life such as providing guidance, transporting objects, and security patrolling. *enon* houses an internal storage compartment which the robot can load and

unload autonomously, thus being able to deliver goods without human assistance or supervision. Fujitsu lists a wide range of features: 1. Autonomous navigation using camera-based obstacle detection and relying on a pre-programmed map. For this purpose the robot is equipped with six cameras, three ultrasonic sensors, and three proximity sensors; 2. Transportation of objects with a mass of up to 10 kg; 3. Handling of objects up to 0.5 kg with a 5 DOF arm; 4. Communication with people using speech recognition and synthesis as well as a touch screen GUI; 5. By connecting to a WiFi network the robot can acquire necessary information and receive command; 6. Using a swivel head the the robot can either look at the operator while communicating or in the driving direction while moving without turning the robot (and the mounted display); 7. *enon* uses LED incorporated into the face to show facial expressions; an finally 8. “Fujitsu has placed utmost priority on making *enon* safe, incorporating a variety of safety features including significantly reducing the weight and width of the robot compared to its prototype”.

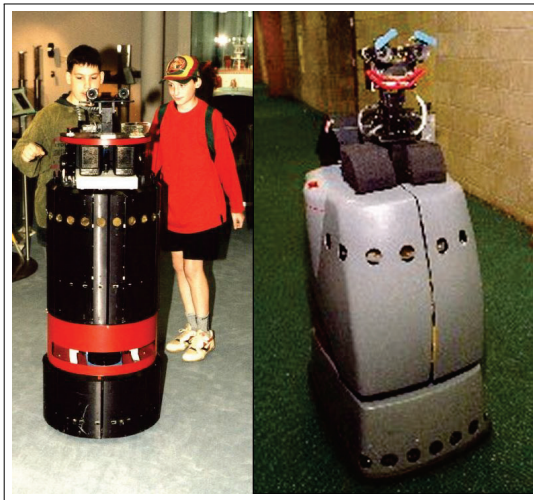


Fig. 2.21.: Rhino (image source: [172]) and MINERVA (image source: [135])



Fig. 2.22.: Robox at Expo02 (image source: [42])

### 2.4.3. Shopping assistants and shopping robots

As shopping in the supermarket is used as application scenario of this thesis a complete section is dedicated to the various shopping assistants found in shops and laboratories. A variety of shopping assistant system, ranging from small handheld devices via enhanced shopping carts to full-scale *service robots* will be introduced. Many projects focus on handheld devices or even smartphone Apps (e.g. Metro, EDEKA) because they are reasonably cheap and easy to roll out. These are the projects most commonly found in stores and actually in commercial use. A more sophisticated version are enhanced shopping carts where more sensors and computational power can be carried to enable for example self-localization. And finally systems like *Toomas* and *Robovie* are full scale *service robots*.

#### Handheld devices:

The most basic type of shopping assistant are handheld devices. They are cheap and easy to integrate in the shops' infrastructure – often a WiFi network is sufficient. Due to their small size they cannot include many sensors. For the users' convenience, many of them can be attached to shopping carts, bridging to the second



Fig. 2.23.: Care-O-Bot of Fraunhofer IPA (image source: [49])



Fig. 2.24.: *enon*, developed by Fujitsu Frontech [52]

category – the enhanced shopping carts. The continuous improvement of smart-phones is giving this area additional drive: shopping assistant Apps are conquering the market. They are popular with shop owners because no extra hardware is necessary and with customers as they can use their own device. The drawback is that smartphones cannot be mounted on shopping carts and hence cannot connect to the shopping cart's sensors. Additionally, the shopper has to carry the phone in one hand during navigation.

**Metro MEA App:** The *Mobiler Einkaufsassistent* (MEA) smartphone App ([114], Fig. 2.25) is developed by the Metro Group's Future Store Initiative ([113]). It provides a mobile shopping list and offers product information when scanning a bar code with the mobile phone's camera. And finally, the App can be used to pay the scanned products.

**EDEKA App:** The *EDEKA App* ([40], Fig. 2.26), developed by Burda Digital Systems GmbH for Edeka Südwest, implements a shopping list and recipe management including the *Food Shaker*: customers enter some products and shakes the smart-phone. The App then generates a matching recipe.

**Giving Cart:** A Handheld device developed by Klever Marketing and Timedomain which can be attached to ordinary shopping carts ([94], Fig. 2.27). It uses WiFi localisation and provides a bar code scanner. The customers can use it to scan items in order to check the price. A store directory is available to assist locating items. The App can automatically generate a shopping list based on individual consumer's prior history.

### Enhanced shopping carts:

In between of the already introduced group of hand-held devices and the shopping robots, the enhanced shopping carts are located. These combine the functionalities of the hand-held devices with more computational power and extra sensors – often used for self localization – while still being cheap and easy to incorporate. Hence, this is a popular group of systems. The enhanced shopping trolley applications



Fig. 2.25.: The *Mobiler Einkaufsassistent* (MEA) smartphone App developed by the Metro Group (image source: [8])



Fig. 2.26.: The *EDEKA App*, developed by Burda Digital Systems GmbH for Edeka Südwest (image source: [7])



Fig. 2.27.: The *Giving Cart* by Klever Marketing (image source: [131])

introduced below have many similarities to the enhanced shopping trolley *ETrolley* (see Fig. 2.33 and Appendix B) constructed in the course of this thesis. But additionally, *ETrolley* is equipped with the complete navigation and self-localisation system including sensor systems, embedded PC and a larger touch screen interface.

**Concierge for Cart:** The *Concierge for Cart* (Fig. 2.28, [111]) developed by Mercatus (former known as Springboard) incorporates a touch screen and a bar code scanner and acts as navigation system and mobile cashier. All gathered information on the shopping trip is fed into the shop's *business intelligence system* to enable personalized and multi channel advertising as well as to generate a better understanding of the customers' behavior. Being able to address the customers' need in a better way shall lead to a higher degree of customer loyalty.

**U-Scan:** The *U-Scan Shopper* (Fig. 2.29, [53]) was designed by Fujitsu and Klever Marketing to reduce checkout queues. A computer with WiFi network access is mounted on a cart that provides information on products and enables the users to scan their products while shopping and to check themselves out. The system can also be used to place in-shop orders i.e. to the deli or the pharmacy.

**Smart Cart:** DFKI and Globus are driving the *Innovative Retail Lab (IRL)*, developing a variety of concepts to ease shopping and related tasks. The *Smart Cart* (Fig. 2.30, [38]) incorporates a personalized shopping list management and navigation functionalities. Using RFID tags on the products, the system is able to detect the products which are place into the basket without scanning the bar code. Precise self-localisation can be performed by different methods such as RFID tags embedded into the floor.

**KleverKart:** The *KleverKart* (Fig. 2.31, [94]) developed also by Klever Marketing, the inventor of the *Giving Cart* handheld device. The system features a self-service assistant with various functions like product information, self-checkout, a recipe database, advertisements based on localization data, and so forth. According to the authors "*KleverKart* has proven in trials to significantly increase the amount of money spent in a store".

## 2. State of the Art

**IBM Personal Store Assistant:** The *Personal Store Assistant* (Fig. 2.32, [79], webpage: [80]) was developed by IBM Retail Store Solutions and IBM Research Services. It is mounted on the handlebars of shopping carts and is connected to the *IBM Store Integration Framework* by Wifi. The framework incorporates a data management system which manages personal information like past shopping trips. The system is able to localise itself using ceiling-mounted infrared beacons. The systems can additionally be used to place orders to the deli counter of the shop and as self-checkout device.



Fig. 2.28.: The *Concierge for Cart*, developed by Mercatus [111]



Fig. 2.29.: The *U-Scan Shopper*, developed by Fujitsu (image source: [54])



Fig. 2.30.: The *Smart Cart* by DFKI's *Innovative Retail Lab* [38]



Fig. 2.31.: The *KleverKart*, developed by Klever Marketing (image source: [93])



Fig. 2.32.: The *IBM Personal Store Assistant*, developed by IBM Research Solutions [79]



Fig. 2.33.: For comparison: the *Enhanced Trolley (ETrolley)* designed in the course of this thesis

### Shopping service robots:

The final group, the shopping robots, are the highest class of shopping assistants. Most of them incorporate lots of sensors and they are designed to fulfill various tasks. But obviously, they are complex systems and expensive to acquire, install and maintain. The most prominent ones will be introduced in this section along with their technical characteristics and most prominent features.

**Toomas and Shopbot** The robot type *Toomas* (Fig. 2.34, [66], [117]) was developed by the group of H.M. Gross at the University of Ilmenau (Germany) as an interactive shopping assistant and mobile information kiosk. *Toomas* robots have been deployed in three “toom” hardware stores during the years 2007 to 2009. *Toomas* does not serve one dedicated user but assists everyone who approaches it. The robot

provides three kinds of services: product information, video conferences with employees, and guiding to products.

While *Toomas* patrols the corridors, it autonomously looks for customers who might be interested in assistance. For this purpose the panorama picture is analyzed for skin color and motion, the high res picture for faces, and the laser scans for legs. After detecting a customer with interest of assistance the robot approaches the customers, and starts a verbal dialog. The customer is invited to give commands to the robot using the touch screen interface. Voice input is not included due to the high level of background noise. The authors point out that employees and customers are very satisfied with *Toomas*: the robot is able to handle about 80% of the customer's requests, being a great relief for the employees. More than 80% of the customers have been "content" and would use the robot again.

*Toomas* is based on the SCITOS-G5 platform [112] which is of an advantageous slender and circular shape and uses a differential drive. The robot was designed for a low need of maintenance and long times of operation without recharge – up to 12 hours. For environmental perception the robot is equipped with laser range finders, a ring of 24 ultrasonic sensors, a high resolution camera system and a panorama camera. The data from this wide range of sensors is fused to overcome failures due to the high complexity of the environment. The fused data is used for obstacle detection as well for detecting and tracking persons in the vicinity of the robot.

*Toomas* uses a three-layered control architecture: (1) The *Hardware Layer* encloses the hardware, the operation system and the low level interfaces; (2) In the *Skill Layer* the low-level sensor information is processed and a set of skills is provided which again are executed by the lowest layer. These skills contain the classical robotic functionalities like obstacle avoidance, localization, person tracing, and so forth. And finally (3) the *Application Layer* where a central state machine controls the communication interface and the high-level behaviors of the robot.

The core of the navigation system is the occupancy map of the store. It is built by a particle filter-based SLAM algorithm while the robot is initially being steered through the shop by joystick. A set of good particles is stored as global map for the global self-localization by *Monte Carlo Localisation (MCL)* later on. The map is then semi-automatically annotated with the product location from the shops management system based on CAD data. Path planning is done using a  $A^*$  algorithm and collision avoidance by the well known *Vector Field Histograms (VFH)* based on distance and visual information.

Since the year 2008 two successors of the type *Toomas*, the *ShopBots* "Ally" and "Roger", are deployed by the company *Metralabs* in the *real,- Future Store*. Here they operate as guides showing the newest inventions of the *Metro Group* to the customers.

**RTS** The *Robotic Transportation System for Shopping Support Services* (Fig. 2.35, [108], [166]) developed by the Toshiba Corporation is an exceptional case. To solve the problem of having an agile and interactive *service robot* which also has to transport a significant amount of goods, the system consists of two individual robots: a slender circular-shaped robot for interacting with, and guiding the user as well as a larger robot for carrying the goods. Consequently, only the "Guidance Robot" is equipped with camera systems, touch sensors and a touch screen interface. It uses an omnidirectional camera for self-localization based on map-matching as well as laser range finders and ultrasonic sensors for navigation which again is based on *Vector Field Histograms (VFH)*. The "Guidance Robot" is able to guide or to follow the user whom it observes with a stereo camera system. The "Cart Robot" just trails some distance behind the

“Guidance Robot”, following its trajectory. The self-localization is performed by measuring the relative position towards the “Guidance Robot”. An environmental multi-camera setup supports the onboard sensors by tracking the robots and the people in the shop. A three-layered control architecture was developed consisting of a self-localization layer, a dynamic path planning layer, and a motion control layer, with the latter one running at 1kHz.

**Robovie** The *Robovie* robot family, [85], [60]) is being developed by the *Advanced Telecommunications Research Institute International (ATR)* (Japan) to do research in human robot interaction. Looking similar to the *enon* robot – a robot with humanoid upper body on a mobile platform. A field trial has been conducted with *Robovie-II* in a grocery store where the robot is carrying goods for a customer by carrying a shopping basket in the crook of its arm. The robot is deliberately designed to be small not to scare people: it is about 120cm high, has 40cm in diameter, and weights about 40kg. It is equipped with two 4 DOF arms, mainly for performing gestures to support the vocal communication. To enable touch-based interaction ATR developed a touch-sensitive skin for the robot. The mobile platform is differential driven and uses an omnidirectional vision sensor as well as 24 ultrasonic sensors for obstacle detection. For the purpose of self-localization and navigation a visual map based on topologically connected omnidirectional pictures is used.

The robots’ control architecture is based on *situated modules*. These modules represent basic actions of the robot such as “Detect Door”, “Turn to door”, “Look at”, “Go forward”, “Go to object”, and so forth. Each module has a set of pre conditions and actions. The task planner of the architecture generates a sequence of modules to be executed based on given start and goal conditions. According to the authors this strictly plan-based approach distinguishes their architecture from the group of *Behavior-Based Controls*, in which proper task planning is very difficult. Special sensory modules are connected to pictures in the visual map. This way for example the “Go to” modules detect that they succeeded and thus are de-activated. The architecture houses two error recovers modules: the first one is the “Module searcher”. This module lets the robot wander randomly until some other module can be activated. The second one is the “Obstacle Avoidance”: when approaching an obstacle this modules navigates the robot into another direction. Another special module is the “Evaluator module”. If several modules can be activated, it evaluates the modules and chooses the best fitting one.

### 2.5. Discussion and resume

This section provides a short overview of the introduced shopping robots and compares them to the robot *InBOT* which is an exemplary implementation of this thesis’s concept. The focus in this section is on abilities and characteristics regarding the supermarket scenario. And finally, the choice of approaches used in this thesis will be motivated.

The robot *Toomas* developed by the group of H.M. Gross ([66]) is an interactive shopping guide robot that patrols the corridors, looking for people which might need assistance. After identifying some in need, it autonomously start an dialogue with the person, which might end up in the robot guiding the user to a products of choice. The robot is primarily focused on interaction and guidance functionalities. In field trials in an hardware store the robot has successfully guided 8600 customers to their chosen goods. Even though the context of the robot is comparable to *InBOT*, the application of *InBOT* is wider: it shall transport goods for customers, extending the amount of possible use cases, resulting in more *modes of operation*. When having a closer look there can be found several differences between the robots. The most obvious one is





Fig. 2.34.: The mobile information kiosk TOOMAS has been operating in a TOOM hardware store. (image source: [167])

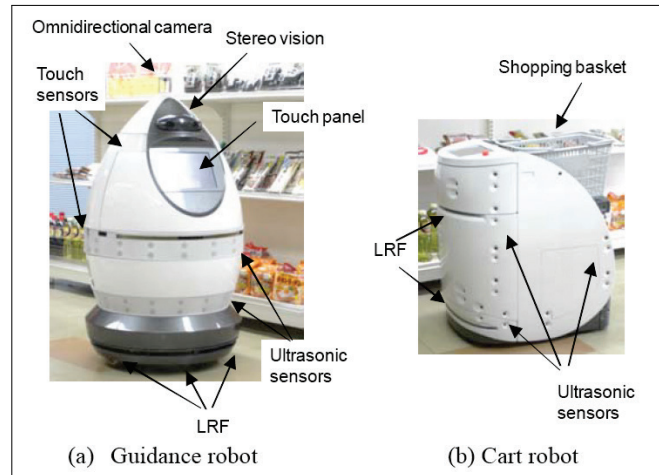


Fig. 2.35.: The *Robotic Transportation System for Shopping Support Services (RTS)* [108] developed by the Toshiba Corporation

robot itself: The platform of *Toomas* (SCITOS-G5) is semi-humanoid with a slender circular base, whereas *InBOT* is intended to carry goods in a basket and hence has a larger rectangular shape which demands more skills from the control system when navigating in cluttered or narrow spaces. While *Toomas* relies on a global occupancy map for navigation and *Vector Field Histograms* for obstacle avoidance, *InBOT* uses a *topological-metrical map* for the global and a local occupancy map for the local navigation. Like the navigation system the obstacle avoidance is implemented in *InBOT*'s as *Behavior-Based Control*.

Kanda's group ([86], [11]) works on an "affective guide robot" in a shopping mall. They already conducted a field trial for 25 days in a shopping mall with 235 participants. The robot has three main functionalities: (1) Guiding customers to products, (2) building up a relationship with customers and (3) advertising products. For customization customer-assigned RFID-tags are used to recognize customers again. This allows the robot to greet known customers and make tailor-made advertisements. In comparison, the robot *InBOT* has less social but additional functional abilities, like following or manual steering behaviors. Kanda's robot is like *Toomas* a semi-humanoid robot. It is based on a mobile platform with the advantageous slender and circular shape. Additionally, it is equipped with arms which are used for pointing gestures, showing a human user the direction to go while in guiding mode. It has speech input and output like the integrated *CR-UI* (see Appx C.5) on *InBOT*, but in contrast it does not have a GUI where shopping lists can be managed. Finally, the authors point out in their publication that they use a human operator for speech recognition and decision making for the robot's behavior. On *InBOT* all these components are integrated in the system and no human operator performs remote control.

Tokura [166] develops the *robotic transportation system for shopping support services*. Unlike other shopping robot projects, this system consists of a pair of robots: a guidance robot and a cart robot, which increases the costs and the space required. The guidance robot has only a touch screen, in contrast to the multimodal user interface integrated in *InBOT* like the *CR-UI* or the *InBOT-UI* (see Appx C.4). The system has mainly the two functionalities: guiding and following the user.

All mentioned systems but *Robox* and *Teletift* navigate based on a detailed global metrical map. These two exceptions move on a defined graph; *Robox* is able to temporarily leave the graph to move around blocking

obstacles. *Telelift* again uses a global geometric map for self-localization via map-matching. Sometimes the map has to be generated manually, but in most cases it is learned by the robot semi or fully autonomous. The concept presented in this thesis is designed to be independent of detailed global metrical map. It is sufficient to only know a local metrical map in the sensor-range of the robot. This way the robot is independent from changes in the environment such as placing or removing of special offer counters or larger numbers of parking (ordinary) shopping carts. Not having a global map means that map matching algorithms for self-localization can not be applied. But out of experience these algorithms are subject to disturbances when the geometry of the environment is changed by placing or parking many objects (such as ordinary shopping carts) at the sides of corridors. Here the corridor suddenly seems to be too narrow and cannot be matched with the map correctly, resulting in a loss of accuracy. Instead, this thesis proposes using landmarks which can be robustly detected and precisely localized: ground-mounted RFID barriers which are unique, easy to identify and which cannot be occluded by any means (in comparison for example with vision-based methods).

The collision avoidance is performed by a *Behavior-Based Control* in an hierarchical three-stage manner: safety behaviors, reactive behaviors based on virtual force vectors, and predictive behaviors similar to local visibility methods. The methods implemented inside the individual behavior have some similarities with the well known *VFF* and *VFH* methods, but in detail there are major differences as will be explained in the introduction of the next chapter. The challenge of avoiding moving obstacles will be again tackled in a three-stage approach. Two levels of reactive behaviors again provide virtual force vectors. These are supported by a spatio-temporal planner in the upper part of the system. Using this strictly hierarchical approach throughout the architecture, the fast reaction time and robustness of the reactive methods can be fused with the capabilities of the plan-based approaches. Actually, in each level a simple dedicated language for all involved behaviors is defined to ease inter-behavior communication and coordination.

To the author's knowledge there are no social *service robots* controlled by a *behavior-based* control system in applications involving closely coupled user interaction like museum guides or intelligent shopping carts. Robots using a *Behavior-Based Control* tend to be very application specific instead and don't follow the needed general-purpose approach needed for the real world application mentioned above. Up to now, they are only found in laboratory or office environments (e.g. *ARTOS*) and sometimes in outdoor environments (e.g. *RAVON*) with the main application of exploration.

*InBOT* did not make the final step into a real shop like for example *Toomas* did, but its development was clearly aimed for this purpose including the needed wide range of functionalities and the needed predictability and robustness of the system. These features and the closely coupled interaction with users – involving even *control sharing* – implemented by a *behavior-based* control system is unique in the present combination on this thesis's robot. To evaluate *InBOT*'s abilities as closely as possible to the scenario without having a huge integration effort, user-tests in a small simulated shop in the laboratory have been performed with untrained users recruited “from the street”.

### 3. The Hybrid Control Architecture

This chapter introduces the control architecture, which was developed in this thesis. The architecture shall handle the challenges and requirements identified in Chapter 1 with a special focus on the interaction between the robot and one dedicated user. The chapter starts with the introduction of the major inspirations for the control system: J.P. Hoogendoorn's model of pedestrians' motion, the *Behavior Networks*, and *control sharing*. Afterwards, design criteria will be defined and the fundamental concepts utilized in the control architecture will be chosen. Finally, the resulting control architecture will be described, including the *local world model*, interfaces, the data flows, and two exemplary implementations in the robots *InBOT* and *Odete*. The navigation system implementing the architecture will be elaborated on in the subsequent chapters.

The scenario chosen as the recurrent theme of this thesis, shopping in a supermarket, addresses several everyday problems posed in supermarkets. The primary challenge posed to the robot is that it has to cope with a highly dynamic environment while simultaneously interacting with a user. This includes the presence of cluttered corridors, moving objects and the fact that the spatial arrangement of the environment changes frequently making it very difficult to rely on a global metrical map.

This plenitude of challenges and the amount of required functionalities demand a capable and well-structured architecture for the robot's control system. Otherwise, one would take the risk of losing a clear view on the control system and would probably get in-deterministic reactions from the running system. Another important aspect for this architecture is the idea of generalizability. Hence, it should be possible to integrate a plenitude of components within the control system. Therefore the architecture has to ease the integration of different components as well as make it easy to substitute temporarily unavailable components for simulations to be able to test the remaining system. This is also the case for hardware components like sensor systems.

#### 3.1. Inspiration for the control architecture

The concept for the control architecture developed in this thesis is mainly inspired by four works: First of all this is the model of pedestrian motion patterns that S.P. Hoogendoorn [73] developed to simulate crowd movements and evacuation situations. This concept was transferred into a control paradigm which then has been merged with the second inspiration: hybrid layered architectures such as the well-known 3-tier (3T) architecture [22] for mobile robots developed at the *Johnson Space Flight Center*. The third inspiring work, the *Behavior Networks* developed by J. Albiez [3], is used to implement the lower layers of the control architecture. And finally the fourth inspiring work is the concept of *control sharing* and *control trading* defined by T.B. Sheridan at the *MIT* which he describes in his book "Telerobotics, Automation and Human supervisory control" [147]. These four works, and their contribution to the control architecture of this thesis, will be presented in the next sections.

#### 3.1.1. Model of pedestrian's motion patterns as blueprint of the navigation system

The navigation in human everyday environments is a very challenging task. When designing the control system it seems logical to have a look at subjects which are already able to operate in such environments perfectly well: human beings. Thus the control architecture developed here is inspired by research on human motion patterns performed by S.P. Hoogendoorn [73] in the context of *Evacuation Dynamics* – a short introduction can be found in Chapter 2.1.

Hoogendoorn divided the behavior of simulated pedestrian agents in three hierarchical layers. These are illustrated in Fig. 3.1 (left) and the navigational purpose of each layer is sketched in Fig. 3.1 (right).

1. *Strategic behaviors*: The topmost layer represents the human route selection based on landmarks. It roughly defines the way that has to be taken to move from the starting point to the goal. A list of nodes containing landmarks or areas that have to be passed/crossed is generated here instead of a continuous path. In pedestrian movement simulation the set of possible nodes is manually defined by the scenario designer. On a robot the implementation could be based on *topological maps*.
2. *Tactical behaviors*: The middle layer refines the given route based on the local environment and infrastructure like stairs, doors, walls or corners. Here again nodes are generated instead of a continuous path. This concept can be directly transferred to a robot's navigation by performing a geometrical analysis of the local environment, placing sub-goals near prominent features like corners. To preserve one of the main advantages – humans are able to operate with only sparse knowledge and local perception – it would be necessary to restrict the robot's scene analysis to the current sensor readings of the robot.
3. *Operative behaviors*: In the bottom part of the hierarchy the real movement behavior is generated. Here the agent moves from node to node until it reaches the target destination. While doing so it adjusts the path to local disturbances like obstacles or other agents. These behaviors could be implemented in a robot using one of the numerous reactive methods for collision avoidance and motion control. Here again it would be important to only rely on local and current information.

The model suggests a three-layered architecture with deliberative components in the top layers and reactive components in the bottom layer. An benefit which very well matches the identified requirements is the fact that the *global navigation* of human beings is based on fuzzy experiences and the local navigation is based on current sensor readings only. Therefore this source of inspiration supports the idea of a hybrid and hierarchical control architecture as has been identified as favorable in the introduction of this thesis.

#### 3.1.2. Hybrid and layered architecture as organizational paradigm

As described in the state of the art, hybrid architectures are very popular as they enable the designer to utilize likewise planners or reasoners for global tasks and reactive components for time-critical actions. In most cases these architectures are organized in form of hierarchical layers to provide structure and to distinguish between different demands on cycle-times and other constraints. Due to the obvious advantages, this fundamental concept can be identified as most prominent organizational policy in the control architecture of this thesis.

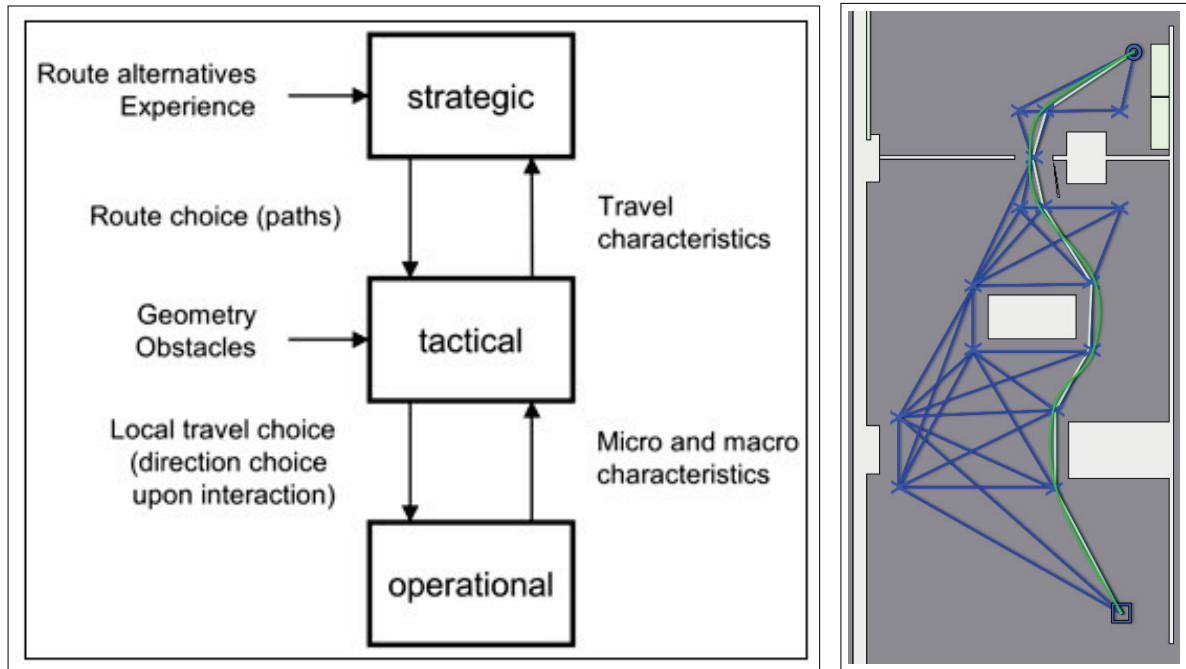


Fig. 3.1.: Left: pedestrian Behavior Hierarchy [73]. Right: again the edge mesh known from the SoA (see Fig. 2.2 for a larger sketch), modeling the pedestrians' motion according to the hierarchy. Strategic: cross the middle room from south to north. Tactical: dark blue edge mesh along local geometry. Operational: actual motion generation (green curve).

Two very prominent examples of such architectures illustrate the concept of creating a symbiosis of planners and reactive skills: the 3-Tier Architecture and the architecture by R. Alami. The *3T Architecture* [22] – where “T” stands for “tier” – has been taken up here as an well known example for hierarchical and hybrid architectures. In the *3T Architecture* a *Sequencing Tier* concatenates the deliberative and the reactive tier by breaking the plans produced by the reasoner down into atomic actions and then activating skills accordingly. Another important implementation of the layered architectures is “An Architecture for Autonomy” by R. Alami [1]. Here a constraint-solver in the topmost layer generates plans to utilize the robot's functions which are interpreted as resources. The executions of the plans is supervised and orchestrated in the middle layer which again utilizes reactive skills of the bottom layer. More details on these two architectures can be found in the corresponding Sections 2.2.4 and 2.2.4 of the state of the art.

### 3.1.3. Behavior Networks for the implementation of reactive behaviors

The *Behavior Networks* developed by Albiez [3] for the low-level control of walking robots have been taken up in this thesis (see also Chapter 2.2.3). They are integrated in the lower layers of the control architecture. They were enhanced to be able to fulfill more deliberative tasks. K. Berns uses very similar approaches for his *iB2C* architecture [125] where the *Behavior Networks* are used in all control levels. As the *iB2C* architecture and the control architecture of this thesis were developed at the same time and both are based on the *Behavior Networks* by Albiez, the lower layers of the control architecture of this thesis and *iB2C* show similarities. But while *iB2C* uses the behavior-based approach throughout the whole architecture for the motion control, navigation, and exploration the approach of this thesis is a hybrid architecture which focuses additionally on dynamic environments and user-interaction. Additionally, the actual implementation

of the *Behavior Network* and the individual behaviors by Berns have hardly any resemblance compared to the human motion patterns inspired approach presented here. (More details on the *BBCs* by J.Albiez and K.Berns can be found in Section 2.2.3 of the state of the art).

#### **3.1.4. The concept of *control sharing* and *control trading* to enhance user interaction**

The area of human factors in robot control is recognized by researcher for quite a long time. Four decades ago, T.B. Sheridan from the MIT – probably one of the authors with the most significant impact – began publishing works on human factors in teleoperation in manipulation tasks, for example on undersea vehicles [148]. He described in the context of *supervisory control* the phenomenon of *control sharing* where an operator (supervisor) exerts control on some *control variables* of an automaton while the automaton itself has control over the remaining *variables*. This contrasts the *control trading* where the user gives a command and the automaton executes the task autonomously having full control on all *control variables* [147]. P. Griffiths and R.B. Gillespie [65] discovered that putting the human and the automaton in a collective control loop improves the task performance: the human is able to concentrate on a given task while the automaton takes care of routine tasks or disturbances which would otherwise draw the attention of the human away from his task. The close coupling of user and robot in actually controlling the robot's actions allows the user to perceive these actions while allowing him to contribute his own intentions simultaneously.

Today this phenomenon becomes even more important as service robots have to perform tasks in close cooperation with humans. Depending on the nature of the task simple *control trading* is not sufficient. To be ready for full collaboration a service robot must be able to behave according to the user's demands while contributing own necessities – therefore to share the control with the human user. While Griffiths and Gillespie chose the obstacle avoidance in a driving simulation as the human's task, in the shopping robot scenario the shopping process itself is the human's task which should not be disturbed by trying to manoeuvre the robot through a cluttered corridor. This control share is provided by the robot (based on the chosen *mode of operation*).

*Control sharing* is not a control architecture itself but more an identified concept while observing human beings. This concept has to be transformed into a control paradigm and merged with the other sources of inspiration. It enables the user and the control system to control the robot simultaneously. The share contributed by the user depends on the current *mode of operation*, the control system's share is always the input of the obstacle avoidance behaviors, and additionally – depending on the current *mode of operation* – the global and/or local navigation. More details on this will be presented when describing the design of the control architecture.

### **3.2. Design of the control architecture**

After describing the sources of inspiration for the concept of the control architecture, general design criteria were developed. Corresponding to the design criteria the fundamental methods to fulfill them were chosen and merged with the concepts derived from the described sources of inspiration. This process is described in the next few sections.

### 3.2.1. Design Criteria of the control architecture

In the application of the interactive shopping trolley obviously the robot has to generate plans, to decide, and to initiate actions online in certain time constrains. Making things worse, control systems in such environments have to be complex, providing lots of functionalities. Nevertheless, the complexity must not get out of hand to ensure deterministic and timely reactions. Different interests and inputs have to be balanced. But, or hence, the robot's control system has to react in a timely fashion and has to provide a constant stream of valid output to always be in charge of the situation. Additionally, the essential parts of the control system must not be affected when other components fail. In 1998, R. Alami defined six design criteria for control systems of autonomous robots, which he applied when designing his "Architecture for Autonomy" (see Chapter 2.2.4 and [1]). Following these proven criteria, the design criteria for the control architecture of this thesis were defined:

1. **Programmability** : The robot shall be able to recombine given functionalities to fulfill new tasks.
2. **Autonomy and Adaptivity** : The robot shall be able to adapt the execution of a given task to the current scene and environmental conditions
3. **Reactivity** : The robot shall be able to react to events within time-bounds while achieving the given goal
4. **Consistent behavior** : The robot's reactions shall always be guided by the objectives given by the task and the control system has to continuously provide valid motor commands
5. **Robustness** : The control architecture shall be able to exploit redundancies in the functions. The different functionalities shall be independent from each other so that a failure of one does not spread through the whole system. It shall be possible to test and validate every function separately.
6. **Extensibility and Modularity** : Adding, adapting and modifying of functionalities shall be supported by the control architecture. Especially recombining present functionalities is an issue that has to be focused on.

In addition to these six criteria for inspired by Alami, a seventh criteria was defined to take the structures needed for the *control sharing* into account:

7. **Orthogonal control data flow** : The control flow in traditional *control trading* applications it top-down from receiving a command via planning to the generation of motor commands. In contrast, to integrate the concept of *control sharing* into the control architecture it must be possible to feed in control data flow orthogonal to the traditional top-down control data flow. The control share exerted by the user during task execution must be fed-in directly into the control mechanisms of the corresponding layers.

### 3.2.2. Fundamental concepts of the control architecture

To address the defined design criteria and the needs derived from the sources of inspiration five fundamental concepts or components were chosen. These are introduced below:

**Hierarchical Architecture:** A three-layered control architecture enables to distinguish between different needs in reactivity and deliberativity ranging from the *topological navigation* down to the *local navigation* and safety behaviors. By distributing modules among individual layers it is ensured that simple and reactive modules are not impaired by slower and more abstract ones as the layers generally apply different cycle times for their components. Additionally, using this separation a higher degree of modularity and robustness is achieved due to the stronger independence of the separated functionalities. Defined interfaces between the layers enable their recombination and reuse as well as the integration of other components and finally feeding in orthogonal control data flow.

**Separated environmental representation:** The *local world model* is kept separated from the remaining control system to allow layer spanning access and data exchange. It is organized in different degrees of abstraction from raw sensor readings up to a topological representation of the complete shopping centre. Defined interfaces enable the integration of alternative methods of data acquisition or processing.

**Hybrid Architecture:** The hybrid character enables the use of *Behavior Networks* in the lower layers and planners in the higher layers of the architecture.

**Behavior-based control system:** The main advantage of the *BBC* used in the lower part of the architecture is the fact that many behavior modules try to achieve a given goal e.g. reaching a location, avoiding obstacles, turn to the user, etc. independently. They are arranged in an hierarchical manner and linked to generated a network of independent modules. Higher tasks are fulfilled by letting higher behaviors control a group of lower behaviors. But even while being controlled by a higher and thus less reactive behavior the lower behaviors do not loose their fast reaction times. This matches to all of the design criteria perfectly well: The different behaviors can be motivated or inhibited to rearrange them, the *reactive behaviors* provide a constant stream of valid control commands, a failure of one behavior does not affect other behaviors. Due to the network character of the lower part of the control system, new behaviors can easily be added by integrating new behavior modules.

**MCA2-Framework:** The C++-based *Modular Controller Architecture* ([143],[45]) framework is used to implement the control software. It supports individual cycle times, modularity, encapsulation and the development of hierarchies. It is completely network transparent, enabling different MCA programs which are running on different computers to communicate with each other just as if they were running on the same computer. Additionally, MCA can run individual parts conforming to real-time constraints. The individual behaviors are implemented in software by *MCA-Modules* which can be organized in *MCA-Groups*. The MCA-modules and MCA-groups are executed in a control-sense-loop with defined cycle times. The connections between the modules are called *edges* which transport the control data flow corresponding to the control cycle. Besides the *edges*, *MCA-blackboards* offer network transparent data storage protected by semaphores.

These five components have in common that they all raise the level of modularity of the resulting control system. As result the modules can be recombined at runtime. This way, available functionalities can be merged at the individual levels of the control system. For example the reactive obstacle avoidance developed for the *local navigation* can easily be used as assistance functionality in the *Manual Steering Mode* as well, even though the robot does not know the goal to which the user is heading.



In the development process these components facilitate a stepwise implementation, integration, and testing. This significantly eases the integration process and enables testing and evaluating with various individual subsystem components, generating a more agile development process.

### 3.2.3. Incorporation of the components

As described earlier, two of the sources of inspiration are not control systems themselves but concepts discovered by observing human beings. These concepts were mapped to a control structure to integrate them with the two remaining sources of inspiration and the fundamental methods derived from the design criteria. The following two paragraphs show how a navigation system based on human motion patterns and *control sharing* was incorporated in a hybrid and layered control architecture. This architecture has, just as Alami's architecture, five overall layers. The middle three layers house the navigation and control system, while the outer two incorporate the platform- and application-specific functions.

### Incorporation of the human motion model

As the human motion model shall provide the primary inspiration, it has to have major impact on the control architecture. Luckily, the model of pedestrian's movement patterns can be directly transferred into a layered architecture concept as shown in Figure 3.2. Of special interest is preserving the ability of human beings to navigate based on fuzzy memories and on current perception only. This matches the identified requirement R3 for the control architecture to be independent from a global metrical map.

According to the layers in Hoogendoorn's model, three layers of control are defined here: the *strategic layer* plans a route based on a *topologic-metrical map*. The route is handed down by sending sub-goals in a sequential order. The *tactical layer* receives the individual sub-goals sequentially and activates behaviors for the geometrical scene analysis including the predictive obstacle avoidance. New sub-goals based on the local geometry of the scene are generated and rated. The best rated sub-goal is then transferred to the *reactive layer*. Here the reactive behaviors are activated which drive the robot towards the sub-goal and ensure that no collision will occur on the path.

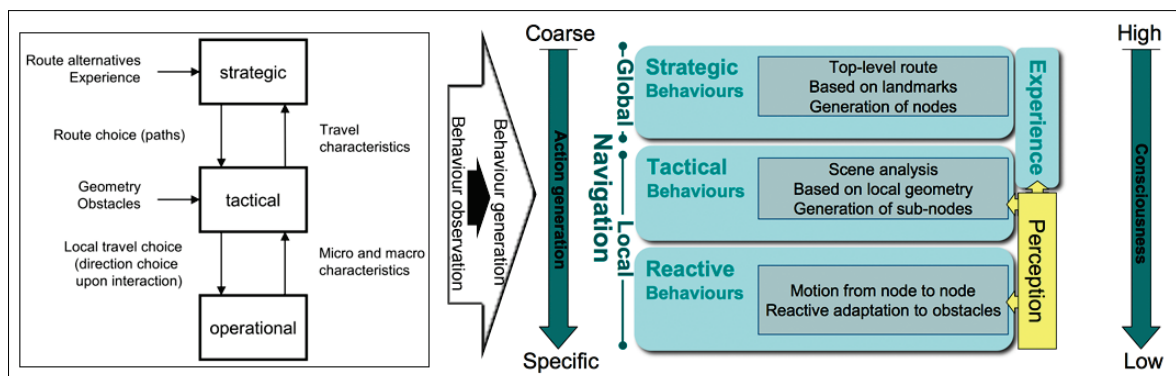


Fig. 3.2.: Transferring Hoogendoorn's motion model of pedestrians into a layered architecture

### Incorporation of *control sharing*

This section describes how the concept of *control sharing* is considered in the design of the control architecture (details and results will be presented in the chapter focusing on human robot interaction: Chapter 6.6 “Sharing and trading of control”). As described before, *control sharing* is not a control architecture and thus has to be mapped on a control system in order to be applied on a robot.

To implement the *control trading*, the control system has to accept commands through a command interface above the *strategic layer*. Responsible for sending these commands is a *communication layer* which belongs to the application-specific part of the control system. The *communication layer* manages the communication with the user and transforms the results into commands which match the list of commands which are defined by the *strategic layer*’s interface. At this point starts the classical top-down data flow.

To implement the *control sharing* the control architecture has to be able to accept control input orthogonally to the usual top-down data flow applied when implementing *control trading*. This input has to be fed into the modules for decision or action generation. Examples for this data is the position, velocity or acceleration of the user or forces the user exerts on a force-sensitive device. Here the most simple way is to use the same interfaces which are used to link the layers to each other (Fig. 3.3). A fusion behavior will then merge the input from the user with the input from the higher layer. This way adaptive behaviors can be implemented which assist the user by taking control of individual components of the navigation system – depending on the current *mode of operation*. Figure 3.4 illustrates this: the safety behaviors are always controlled by the robot. In the *Following Mode* for example the user performs the *global navigation* while the robot performs the *local navigation*. This can be used to implement adaptive guiding or following behaviors as well as obstacle avoidance assistants for the *Manual Steering Mode*.

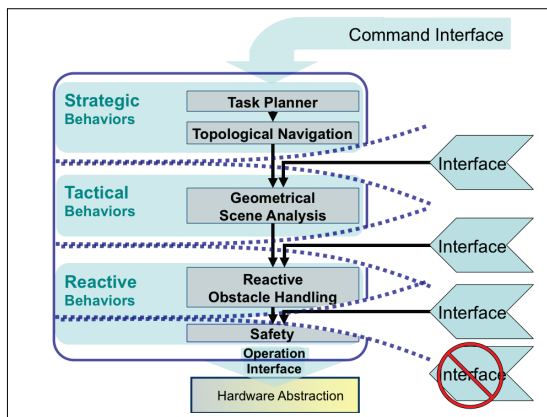


Fig. 3.3.: Inserting control data into the architecture using interfaces between layers. Insertion of user input between the safety behaviors and the *Hardware Abstraction Layer* is not allowed to ensure that the safety behaviors are always in charge.

Modes of operation	Control		
Manual Steering Mode w/o OA			
Manual Steering Mode	Safety	LN: Reactive Part	Global Navigation
Servoing Mode		LN: Tactical Part	Task
Following Mode			
Guiding Mode			
Autonomous Mode			
Control by: robot user			

Fig. 3.4.: *Control sharing* and *modes of operation*: Influence of user and control system on the navigation components depending on the current *mode of operation*.

### 3.3. The resulting control architecture

This section describes the control architecture resulting from the derived requirements, the application of the fundamental methods, and the introduced sources of inspiration. Three main fields of duty can be identified in the demands posed on the robot by the environment and the needed features. These are (1) the task and topological route planning, (2) the handling of the dynamic environment including the interaction with humans, and finally (3) the safe and reliable *local navigation*. Therefore a three-layered navigation concept was foreseen to cover the mentioned three identified fields. The first layer, called *strategic layer*, is responsible for deliberative long-term planning. It houses the *task planner* and the *topological navigation*. The *tactical layer* contains a *behavior-based* control system to deal with moving objects in a local area, with the predictive avoidance of obstacles and with the interaction with nearby humans. The behaviors are based on different modules for *scene analysis* like object and user tracking. And finally, at the bottom of the hierarchy, the *reactive layer's* goal is to fulfil movement tasks inside a local topological area using a *Behavior Network* as its control mechanism. It is based on the dynamic calculation and fusion of several 3D velocity set-point vectors.

These three layers of the navigation system plus the application-specific *communication layer* and the platform-specific *Hardware Abstraction Layer* make up the layered control architecture as illustrated in Fig. 3.5. Details on these layers will be elaborated on in the next chapter when the navigation system is described.

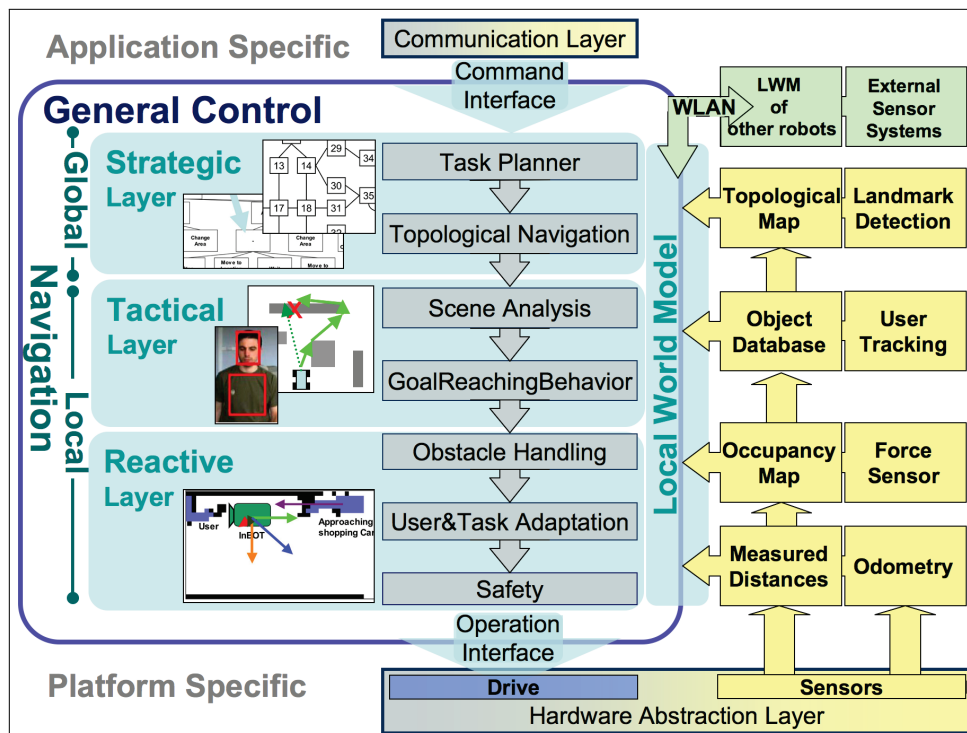


Fig. 3.5.: The robot's control architecture: it consists of the three layers of the navigation system, the *Hardware Abstraction Layer*, the independent *local world model* and a command interface on top.

### 3.3.1. The architecture of the infrastructure

To operate a fleet of robots, these have to be embedded into an infrastructure. The idea is to keep the individual robot as autonomous as possible. Thus they can be introduced more easily and are able to operate even when the infrastructure is temporarily unavailable. Thus the intelligence has to be kept within the individual robots and a central server is only used as central information database.

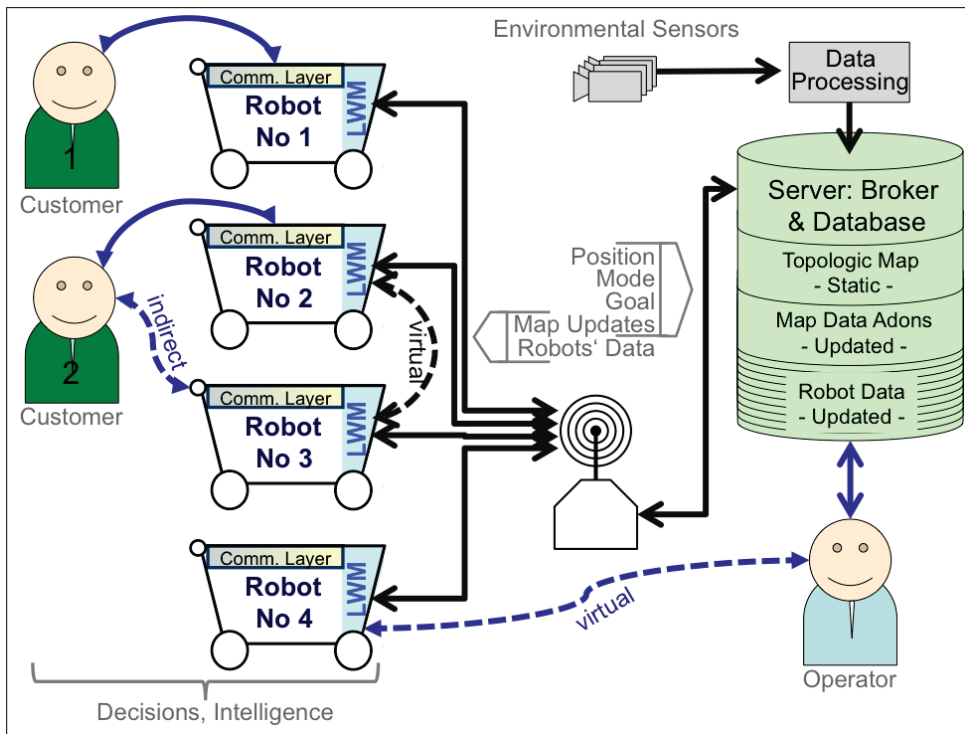


Fig. 3.6.: When operating several robots, these are embedded into a infrastructure. Each robot communicates via WiFi with a central server. The server collects status information such as position and target from all robots and distributes it. Customers can control one robot using the user interface, or a group of robots using the user interface of the master robot. And finally an operator is able to control robots too.

Figure 3.6 illustrates the overall system's setup and infrastructure. When a WiFi connection is established between a robot and the central server, the robot registers itself and continuously sends its current status and updates of the map in case of severe changes. In return the server sends an aggregated collection of the remaining robots' data and map updates, e.g. increased traverse costs for certain topological areas due to heavy traffic. If environmental sensors are integrated (like the *intelligent environment* (see Appendix C.3 "The *intelligent environment*")) the server collects and aggregates and distributes the data, such as the users' positions or the path of other moving objects). A user can control one robot via the user interface provided by the robot's *communication layer*. If the user wants to control more than one robot, the robots exchange the control information via the central server which acts as broker. Finally, an operator logged on the server is to control any of the robots.

### 3.3.2. The *local world model*

According to the control architecture a three-leveled world representation was developed. The *local world model (LWM)* is a layer-independent model of the robots surroundings and inner state. It is divided into

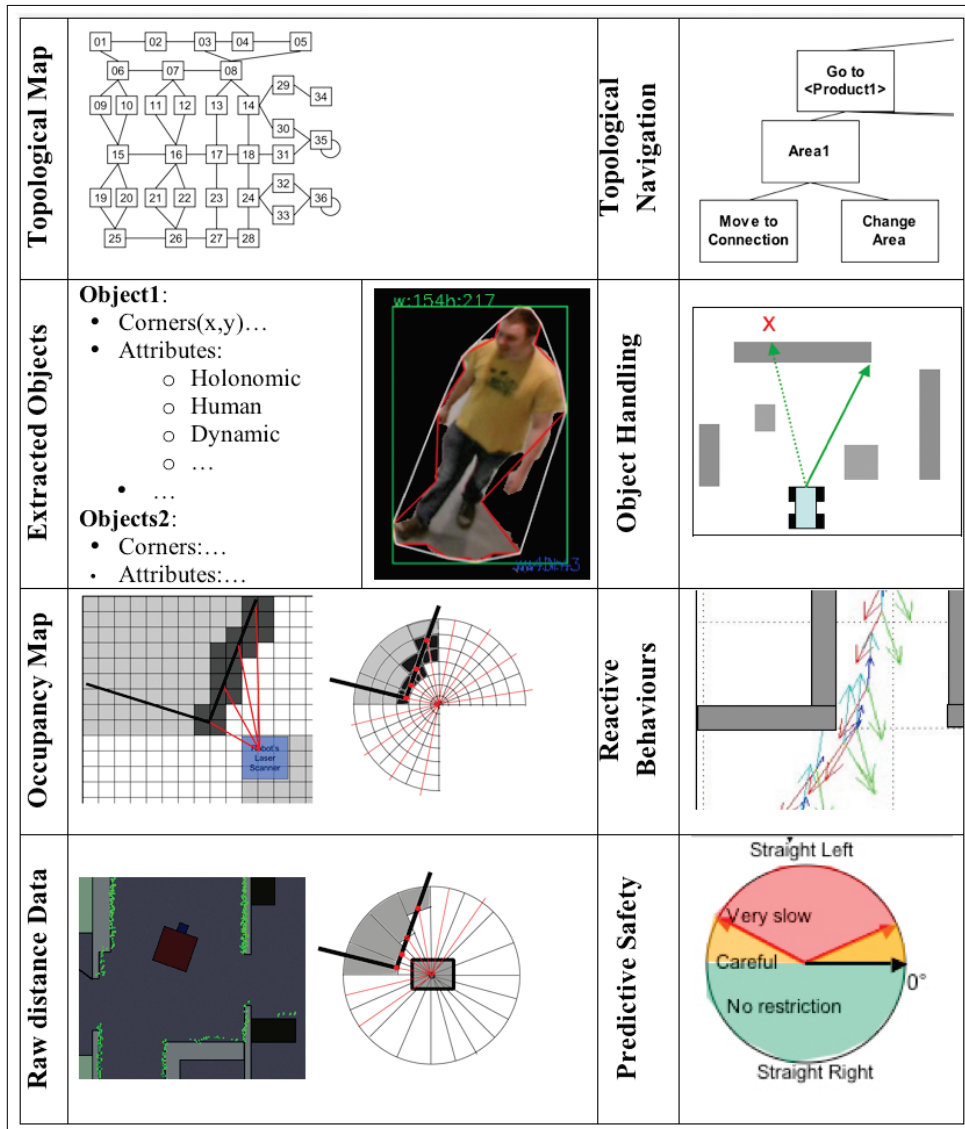


Fig. 3.7.: The four levels of the *local world model* with the corresponding control components.

several levels corresponding to the degree of abstraction of the contained data (see Fig. 3.7). It provides a general interface to pre-processed information accessible for all modules of the control system. It encapsulates the actually used sensor systems and abstracts the information from them, making the control system independent from the sensor hardware just as the *Hardware Abstraction Layer* makes the control system independent from the robotic platform. The information is stored and then shared in a network transparent memory area to be accessible for all components via *MCA blackboards* to ensure high availability of the data. The *LWM* is partially shared between the robots so that these know each other's position, motion, current *mode of operation* and goals. Modifications of the annotated *topological map* are shared as well to inform other robots about structural changes of the environment. The three level of abstraction are introduced here:

1. **Data acquisition level:** The *reactive behaviors* responsible for the obstacle avoidance work based on a binary occupancy grid map that is kept in the world model. These grid merges the information

from various sources like laser range finders whose data is acquired in the *Hardware Abstraction Layer* or information received from other robots or memorized at an earlier point in time. This level additionally houses the inner state of the robot like its position and velocity, the actual task and target or actual errors and so on. Comparable information is kept about the robot's user.

2. **Data interpretation level:** The more deliberative behavior modules make use of an *object database*. This database is generated by both identifying and tracking objects in the grids and estimating their movement models. Additionally, other robots are added when these communicate their position, intended goals, and movements. Finally, the user is identified and tracked. This can be either done by an on-board stereo camera system or an external system – depending on the actually integrated systems.
3. **Abstract data level:** The highest level of abstraction houses the *topologic-metrical map* of the super-market with semantic annotations and online modifications: The environment is split up into several areas. The areas are mapped in a *topological map*. Each node is enhanced with metric information of its size and links as well as with semantic information about the node. The semantic information can be used as an indication of whether a node should be used or avoided in a global plan. Additionally, this information can imply restrictions to velocity or acceleration, certain dangers which wait in this area like a grate in the floor which would render an optical motion sensor useless or “invisible obstacles” like a mirror that could trick a camera system. Additional information like a blockage of a corridor or the prohibition to enter a room or other warnings can be stored. The information can be shared with other robots.

#### 3.3.3. Interfaces

The interfaces between the individual layers and between the layers and the *LWM* are implemented using the MCA framework. There are two types of members of interfaces: *edges* which transport one floating-point value according to the control-sense-cycle of MCA and *blackboards* which represent areas of network transparent shared memory protected by semaphores for the storage of large amount of data which can be accessed from all parts of the control system. Figure 3.8 shows the application of these methods throughout the control architecture.

From a data point of view the interfaces have a defined “language”: topmost these are command and event messages, below we have 3D locations and then finally the velocity set point vectors. Each layer houses a fusion module which collects all the individual tasks in the given language, transforms it to the language of the bordering layer and hands it over.

The components for environmental perception are integrated likewise: the user tracker provides a 3D location of the user and the force sensor provides a 3D force vector which is then interpreted as velocity set point vector.

There is one exception to these MCA-based interfaces. The *command interface* between the *strategic layer* and the *communication layer* is based on a classical TCP connection with a defined set of commands and events. The *command interface* and the reasons for this exception are explained in the next section.

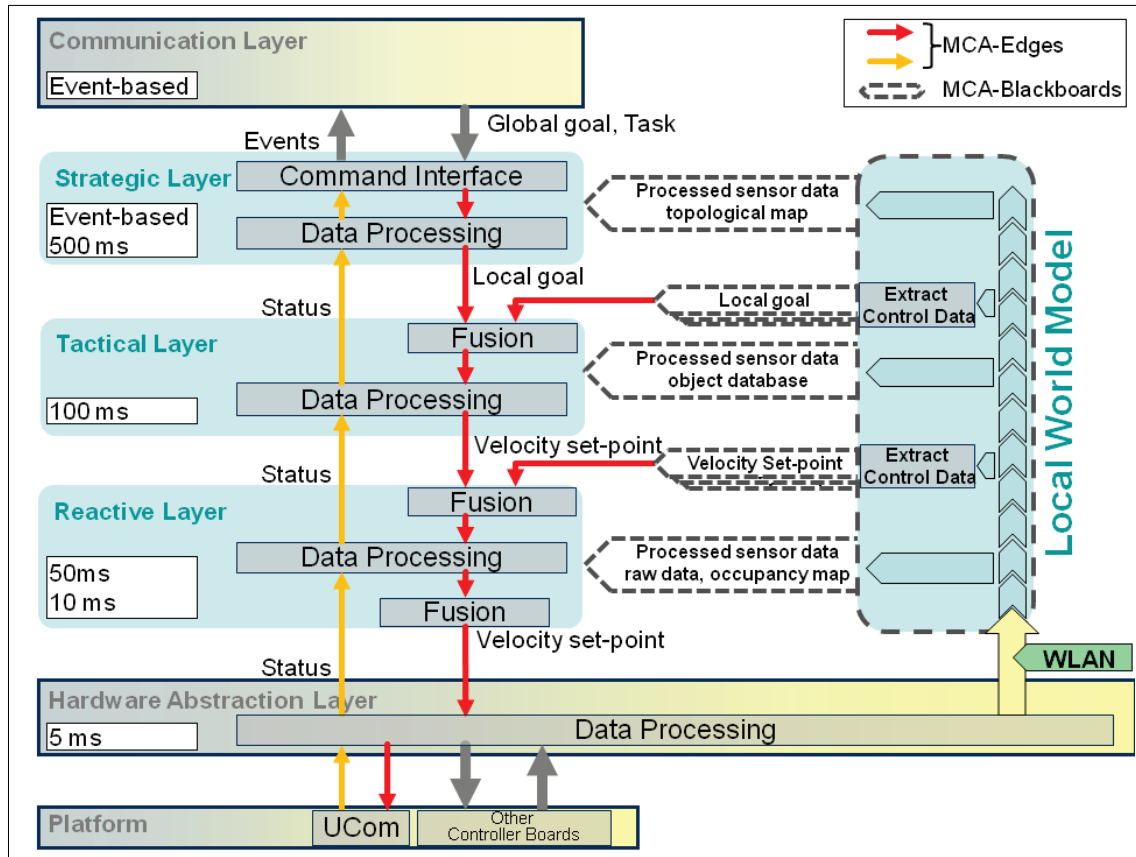


Fig. 3.8.: In the control architecture two control cycles are implemented using the two types of interfaces provided by MCA: the first one connects the control data with the status information using MCA-edges and the second one provides abstracted environmental data using the *LWM* and MCA-blackboards.

### The *command interface*

The *command interface* is implemented by a classical TCP connection instead of the MCA-interfaces applied in the remaining control system. The reason is that MCA is, as the name states, a controller architecture. Thus, it is good for modeling control applications which apply control-sense loops. Other concepts can be implemented in MCA as well, but this is not what MCA was designed for. For the application-specific *communication layer* one might want to use alternative methods and maybe other programming languages instead of C++. As TCP connections are widely spread it was the method of choice. The two exemplary *communication layers* for *InBOT* for example were implemented in QT or Java respectively as they are much more favored in developing graphical high-level applications.

The command messages inform the control system about a task to fulfill (in most cases to move to a certain location) or to switch into another mode of operation. Additional commands include the information if a user is attached or if there is a location of interest nearby where the robot should slow down while passing it.

The event messages inform the *communication layer* about started or finished actions, about problems and if there is a special situation apparent or action planned which could be interesting for the user (“InformUser event”). The information to generate these events is collected from all over the control system, especially from the *local world model*, the target reaching behavior and the *scene analysis* group. The messages are

Tab. 3.1.: *Commands and Events used by the command interface*

Commands	Events
AttachUser	PositionChange
DetachUser	Error
MoveToPosition	ArrivedAt
ChangeMode	LostContactToUser
SetMaximumSpeed	TooLargeDistanceToUser
ShowLocation	UserStopsPushingTrolley
DriveToUser	StartedShowLocation
Servoing	StoppedShowLocation
SetTargetObject	Parking
	BumperPressed
	SendGoal
	InformUser

listed in table 3.1.

Each message is accompanied by a set of parameters, a defined start and end sequence as well as a checksum to ensure that the messages have been received completely and correctly, especially when transmitting via WiFi.

### 3.3.4. Data flow

Figure 3.8 shows the data flow throughout the control architecture. There are two main direction of data flow: control data flows downwards while becoming more concrete and sensor data rises up while becoming more abstract. There are two methods of transporting the information depending on their character: time critical and sparse information are transported directly using MCA-edges, large amounts of information and not time critical ones are exchanged using MCA-backboards via the *LWM*. The two main currents generate several cycles: each layer has its own cycle with a defined cycle time and is connected to the neighboring cycles by the main currents.

### 3.3.5. Integration with the MCA2 framework using *UComs*

The *Modular Controller Architecture MCA* ([143], [168]) was developed at the FZI to be able to set up modular architectures efficiently. As described, the modules are integrated in a continuous sense-control loop. To extend the control (software) system beyond the level of the embedded PCs, special microprocessor boards were developed at the FZI. The UCOM-board ([130]) which contains a DSP with dedicated memory, a FPGA, various A/D and D/A converters, and a CAN-bus interface is integrated seamlessly as the lowest part of the architecture. So called *Remote Parts* can be executed on the PC which let the *UCom* appear to be just an ordinary MCA program or module to the *MCA* framework.

## 3.4. Exemplary system architectures

Concluding this chapter, two applications of the control architecture will be demonstrated. Here the architectures of the robot *InBOT* which was developed in the course of this thesis and the robot *Odete* on which the control architecture was implemented as well will be opposed to each other. These two robots have



significant differences in their hardware and their application, which again result in differences in the architecture. For more details the reader is referred to the Appendix B where the design of *InBOT* is described in more detail.

In Table 3.2 a short overview of the implementation of the control architecture on two robots – the transport robot *Odete* and the shopping robot *InBOT* – is provided. As *InBOT* was developed in the course of this thesis all components, including the application- and platform-specific ones are proprietary developments. For the purpose of evaluation, additional components developed by other researchers were integrated in the robot. In contrast, on *Odete* already present components were integrated with the core layers of the control architecture.

### 3.5. Summary of the control architecture

The designed control architecture is aimed at tackling the challenges posed to a *service robot* operating in a dynamic environment, involving user- and multi-robot interaction. Humans are able to behave in such circumstances perfectly well, thus, the major source of inspiration chosen here is the model of pedestrians' motion patterns by J.P. Hoogendoorn. It proposes a three layer approach using a *strategic layer*, a *tactical layer*, and a *reactive layer*. As the general control system shall not be tailor-made for one specific robot in one application, the three general layers are accompanied by the platform-specific *Hardware Abstraction Layer* and the application-specific *communication layer*. For example on *InBOT* it contains the product database, the multimodal user interface, and the shopping list management system.

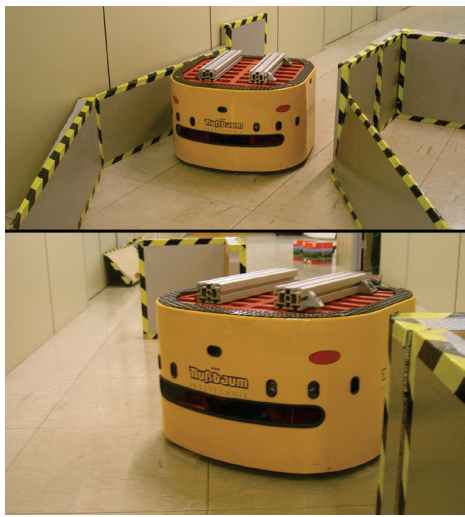

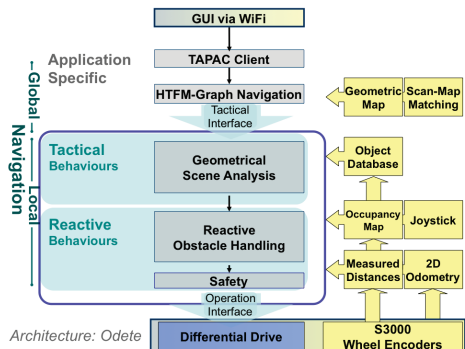
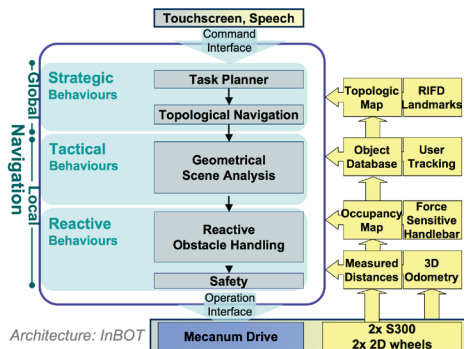
The *strategic layer* consists of a task and route planner, which utilizes a *topological navigation* in order for it to be independent from a global metrical map. The *tactical* and *reactive layers* are implemented by *Behavior Networks* based on work by J. Albiez. These contain behavior modules performing a geometrical *scene analysis* providing for reactive obstacle avoidance. The advantage of the *Behavior Networks* is that a multitude of independent capabilities are fused, instead of developing one monolithic and “omnipotent” path planner which has to consider all factors of influence from the dynamic environment. This way the overall behavior can be adjusted much more easily by parameterizing the already present, or by “hooking in” new behavior modules.

Orthogonal to the common top-down control data flow, user input can be fed into the control system at all layers. This way the control system is able to share the actual control of the robot's behavior with the robot's human user. The control system provides for five *modes of operation* with different levels of autonomy or cooperation: manual steering, servoing, following, guiding, and autonomous operation.

Following the design of the control architecture in this chapter, the next chapters will now focus on the implementation of the individual layers in general and on the behavior modules in particular.

3. The Hybrid Control Architecture

Tab. 3.2.: System architectures of the robots *Odeto* and *InBOT*

	<i>Odeto</i>	<i>InBOT</i>
Appl.	Tele-operated transport	Shopping assistance
The robot		
Control architecture		
Application specific layer	GUI on PCs via WiFi*	Multimodal communication platform incl. speech and touch screen I/O, two different versions integrated (Qt-based <sup>+</sup> , Java-based*)
Strategic layer	Graph-based navigation*	Topological navigation <sup>+</sup>
Tactical and reactive layer	Behavior networks <sup>+</sup> : geometrical scene analysis, predictive obstacle avoidance, based on occupancy map	Behavior networks <sup>+</sup> : geometrical scene analysis, predictive obstacle avoidance, based on occupancy map
Drive	Differential*	Mecanum*
Global self-loc.	Scan-map-matching*	Ground-mounted RFID barriers*
Local self-loc.	Encoders on drive wheels*	Two 2D measurement wheels <sup>+</sup>
Manual control	Joypad*	Force sensitive handle <sup>+</sup>
	<sup>+</sup> : Proprietarily developed components <sup>*</sup> : Other components which were integrated for evaluation purposes	

## 4. BBC: Navigation, Obstacle Avoidance and Safety

This chapter presents the navigation system developed in this thesis. It mainly consists of the *behavior-based* control system. The *Behavior-Based Control (BBC)* includes the individual behaviors implementing the navigation system, such as the local navigation and the collision avoidance. To enhance readability, the behaviors for the avoidance of moving obstacles and for multi-robot coordination will be presented in the two dedicated chapters following the present one. Sticking to the concept of the human motion patterns, the individual layers and behaviors developed here are – just as the overall control architecture itself – inspired by the motion behaviors of human beings. Supporting the *BBC* with high-level planning and navigation abilities, the *topological navigation* and the planning is also described.

**Scope and organization of this chapter:** The chapter starts with the first section describing the the control architecture’s individual layers. This time the description focuses on how to implement these layers with software modules which again follow the presented concept of the human motion patterns. To enhance the understandability of this work, this is done in top-down order, following the control data flow. Later on, the more detailed descriptions of individual components are presented in an bottom-up order.

Following this first introductory section, the next two sections focus on the main components of the navigation system: Section 4.2 summarizes the concept used for the self-localization and Section 4.3 describes the *behavior-based* control system and the *Behavior Network*. Afterwards, the next four sections describe the main components of the four individual layers: Section 4.4 introduces the *reactive behaviors* for safety and obstacle avoidance, Section 4.5 describes the *tactical behaviors* which perform the geometrical *scene analysis* Section 4.6 introduces the *topological navigation* and planning, and finally Section 4.7 will give a brief summary of the application-specific part of the control system, the *communication layer*. Each of these sections contains a paragraph which presents experimental results for the most important components.

The chapter concludes in Section 4.8 with presenting tests conducted with “real users”, followed in Section 4.9 by a discussion on the implemented methods including a brief comparison of the developed methods with the state of the art.

### 4.1. Navigation system and control architecture

In the previous chapter the concept of the control architecture – and the reasons to choose this very concept – was introduced. This section will continue by elaborating on the behaviors developed to flesh out the control architecture – filling the control architecture with live. Just as the architecture is inspired by the observed human motion patterns, the individual behaviors will also follow this concept.

In this section, the basic concepts of the individual methods and behaviors will be derived in an top-down manner, following the control data flow starting at the *communication layer* down to the *Hardware Abstraction Layer*. Functions and implementation details on the individual methods and behaviors can be found in the dedicated sections following the current one.

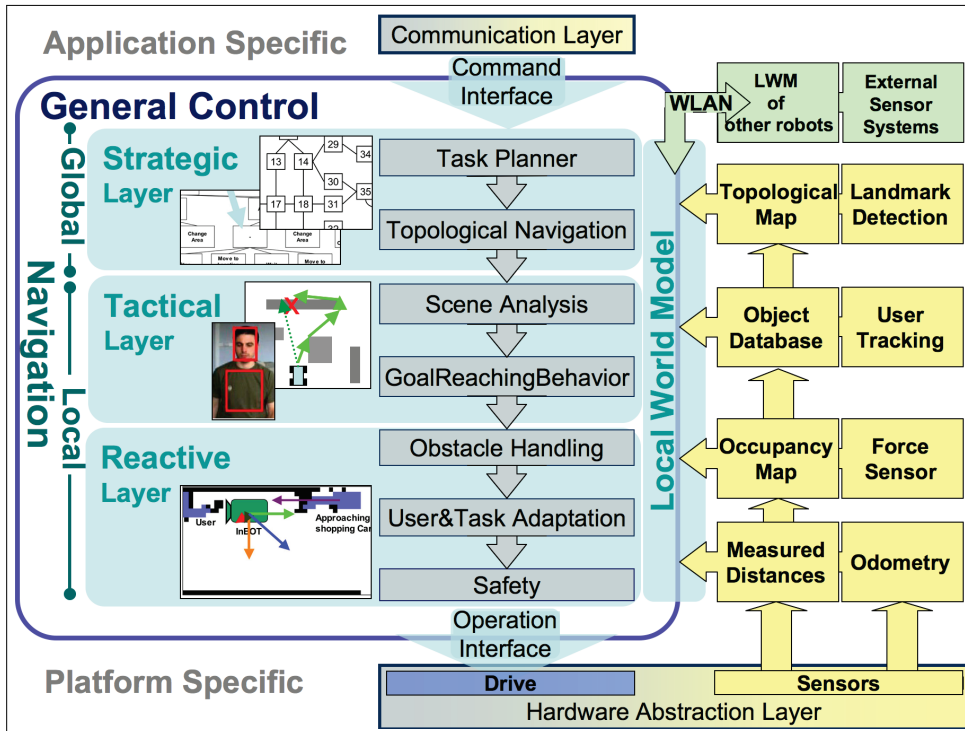


Fig. 4.1.: Behavior hierarchy of the control system: The *strategical behaviors* of the *global navigation* work based on a *topological map*. The *tactical behaviors* and the *reactive behaviors* of the *local navigation* use the *object database*, the *occupancy grid*, or raw sensor values, respectively.

In the introduction of this thesis it was mentioned that one of the goals is the application of service robots in human everyday environments. This requires a safe and reliable navigation system. Further more, the navigation has to be capable of coping with the highly dynamic environment while performing tasks and interacting with a user. As we find a gap in the reaction times demanded for local and global (navigation-) tasks, it makes sense to split the navigation system up into this two components: the *global navigation* part with the task planner and *topological navigation* as well as the *local navigation* and interaction part with the tactical and the reactive behaviors (see Fig. 4.1).

Figure 4.2 roughly sketches the control data flow through the navigation and control system. After receiving a command through the interface from the *communication layer* first off all the global planning, *topological navigation*, and route planning takes place. Here the commands of the user are analyzed and processed. The output of this process is a plan organized in a tree structure containing actions on a topological level including topological navigation points extracted from the *topologic-metrical map*.

The second part of the navigation system, the *local navigation*, contains a *Behavior Network* that fulfills the given tasks inside a local area. Here the two *behavior repertoires* are set up. First the actual scene is observed and geometrically analyzed. Then sub-targets are calculated to generate a suitable route around objects that block the direct line to the target and to avoid trapping situations. Finally, the *Behavior Network* drives the robot along this route and is responsible for the micro-management of the reactive obstacle avoidance and the velocity control.

A *topologic-metrical map* provides for the data needed by the global part and occupancy maps acquired by laser range finders for the data needed by the local part of the navigation. The main idea here is –

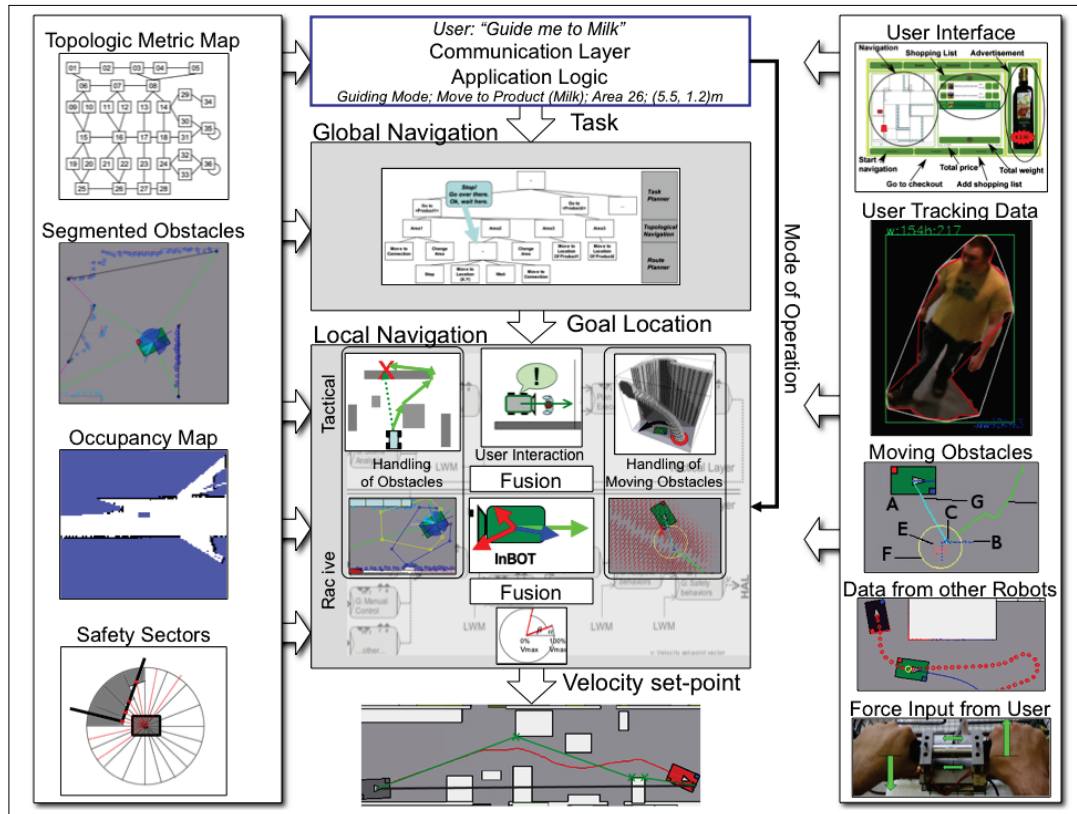


Fig. 4.2.: The control data flow: This figure shows the main components of the navigation system and their integration. The left and right hand side column show sources for data input, the middle column the processing tool-chain.

according to the goals and requirements – to make the navigation independent from a global metrical map because it can be assumed that it would be outdated too fast to rely on it. In contrast, the topological model of the environment should be as far as possible time-invariant. The local part of the navigation falls back to Brook’s motto from 1991: “the world is its best model”.

Using a combination of reactive and deliberative behavior modules based on local online-data on the one hand as well as classical planners and abstract global information on the other hand, a combination of very fast reactions and intelligent path generation and task fulfillment can be achieved.

#### 4.1.1. The *communication layer*

The *communication layer* is responsible for the conscious part of the interaction of robot and user, for the high-level decisions, and for the application-specific task processing. As demanded by the requirement R3 “Abstract and reliable environmental representation”, this layer only uses abstract and application specific information. In the supermarket scenario this is a database of the shop’s products, customer specific information like shopping lists, and complementary information such as current special offers.

Technically speaking, this layer houses the application-specific components. It is used to communicate with the user, which means that it receives commands from the user and communicates events to the user. Depending on the actual application, various modalities for the act of communication can be provided,

ranging from speech in- and out-put to devices like bar code scanners. Input given to the robot is processed using an application logic, resulting in commands for the robot's control system like a movement command or switching the current *mode of operation*. These commands are transferred to the *strategic layer* via a defined TCP-based interface. As described in the previous chapter, the TCP-interface was chosen in order to be independent of certain programming languages, therefore to ease the integration of alternative *communication layers* with the core control components.

##### 4.1.2. The *strategic layer*

The *strategic layer* performs the subconscious but still deliberative decision making of the system. As demanded by the requirement R3 "Abstract and reliable environmental representation", this layer uses abstract and mostly time-invariant information only: a *topologic-metrical map* where the metrical information are strictly bound to static features of the environment, resembling abstract and learned fuzzy information.

From a technical point of view, this layer is responsible for tasks management and the global navigation e.g. the route planning. It accepts commands through the *command interface* as was described in Chapter 3.3.3 "The *command interface*". The received command is then planned based on the topological part of the *topologic-metrical map* of the environment. The manually created *topologic-metrical map* used here can easily be substituted by other global maps like the semantic map by K. Uhl [169] which can be generated automatically as was demonstrated by J. Oberlaender [120].

The plan is refined step wise and stored a tree structure by the global route planner. The tree structure was chosen as it enables the step wise refinement of the tasks as well as interruptions and even postponement of complete sub-tasks i.e. branches of the tree. In the last step the currently relevant leaves of the planning tree are extended with metrical information: the topological navigation points extracted from the metrical extensions of the *topologic-metrical map* or product locations are extracted from the command originating from the *communication layer*. The resulting plan is handed down to the *tactical layer* step wise in form of local goal coordinates  $(X, Y)$  and an optional orientation  $\theta$ .

##### 4.1.3. The *tactical layer*

According to Hoogendoorn's *tactical behaviors* of pedestrians, the main task of the *tactical layer* is to perform a geometrical *scene analysis* to identify objects which either impair the navigation or pose a threat to the robot. Additionally, the user's position and motion are tracked here.

This layer is implemented by the top half of the *Behavior Network* as is shown in Figure 4.2. The *Behavior Network* is designed to fulfill a movement action commanded by the *strategic layer*. In the case of a target location that is to reach, first a predictive obstacle handling takes place which generates new sub-goals. Afterwards a goal-attraction behavior module takes the current sub-goal and generates a corresponding 3D velocity set-point vector which moves the robot towards the sub-goal. This vector consist of two translatory components  $(X, Y)$  and can contain a rotation velocity. Finally, the vector is handed down to the *reactive layer*.

**Predictive obstacle handling:** If a target location location is known, therefore the task is a point-to-point movement instead of for example the manual control of the robot, the predictive obstacle handling takes place to prevent getting stuck in local minima and to generate more efficient paths.

The introduced motion model of pedestrians suggests that pedestrians align themselves to a virtual mesh which connects nodes placed in a comfortable distance to the corners of obstacles as sketched in Figure 4.3 (top). To take the requirements defined in the introduction into account, this layer is only allowed to operate on local data. Thus, generating a global mesh as depicted is not feasible here. The global method has to be transformed into a local one. The straight-forward approach is to only consider the corners which can be currently seen by the robot's sensors.

As benefit of inspiring the resulting trajectory by a visibility graph a more efficient and shorter path is generated. This concept is sketched in Figure 4.3 (left) while Figure 4.3 (right) illustrates that this concept enables the robot to avoid, or even to move out of, U-shaped obstacles. As only local data is available, no path planning can be performed on the local visibility graph – the most beneficial node has to be chosen by heuristics.

#### **4.1.4. The *reactive layer***

In this layer the (sub-)goal-attracting vector received from the *tactical layer* is merged with attracting and repelling vectors added by several dedicated behavior modules to adapt the robot's behavior to the local scene. As demanded by Hoogendoorn's model of pedestrian movements, purely local and reactive methods are applied here. Each individual behavior module generates one velocity set-point vector which represents the behavior's wish regarding the robot's action. In the simulation of pedestrians, in this layer potential fields are applied. But in the present control system much more factors of influence have to be taken into account. Calculating a complete potential field would be rather computational expensive. In contrast, every behavior module calculates one vector according to the specific current situation only – incorporating the current position, orientation and velocity of the robot, of obstacles, and of the user, as well as input provided by cameras, the *force sensitive handle*, and so forth.

The processing chain of the *reactive layer* starts with the task oriented input vectors which are for example received from the *tactical layer*, generated by a module responsible for person following, or acquired by the input of the robot trolley's *force sensitive handle*. These attracting vectors are merged in a maximum fusion behavior module, which means that one data source is selected. The resulting vector is handed to the behavior modules for the reactive avoidance of obstacles. These are grouped in two sub-networks or groups. The group for the avoidance of static obstacles and the group for the reactive handling of moving objects. Each group provides one additional set-point vector which are merged with the task-oriented one in a weighted fusion behavior module. The data basis for this level is provided by an occupancy map accessed inside the *LWM*.

After altering the task-oriented velocity set-point vector in such a way that the robot moves around obstacles, this altered vector again is handed down to the group of safety behaviors. These work on raw sensor distance information and slow the robot down, or – in worst case – stop the robot just before a collision occurs. These behaviors do not change the direction of the velocity set-point vector but they limit its length – down to zero if necessary. After passing the safety behaviors the resulting vector is passed down to the *HAL*.

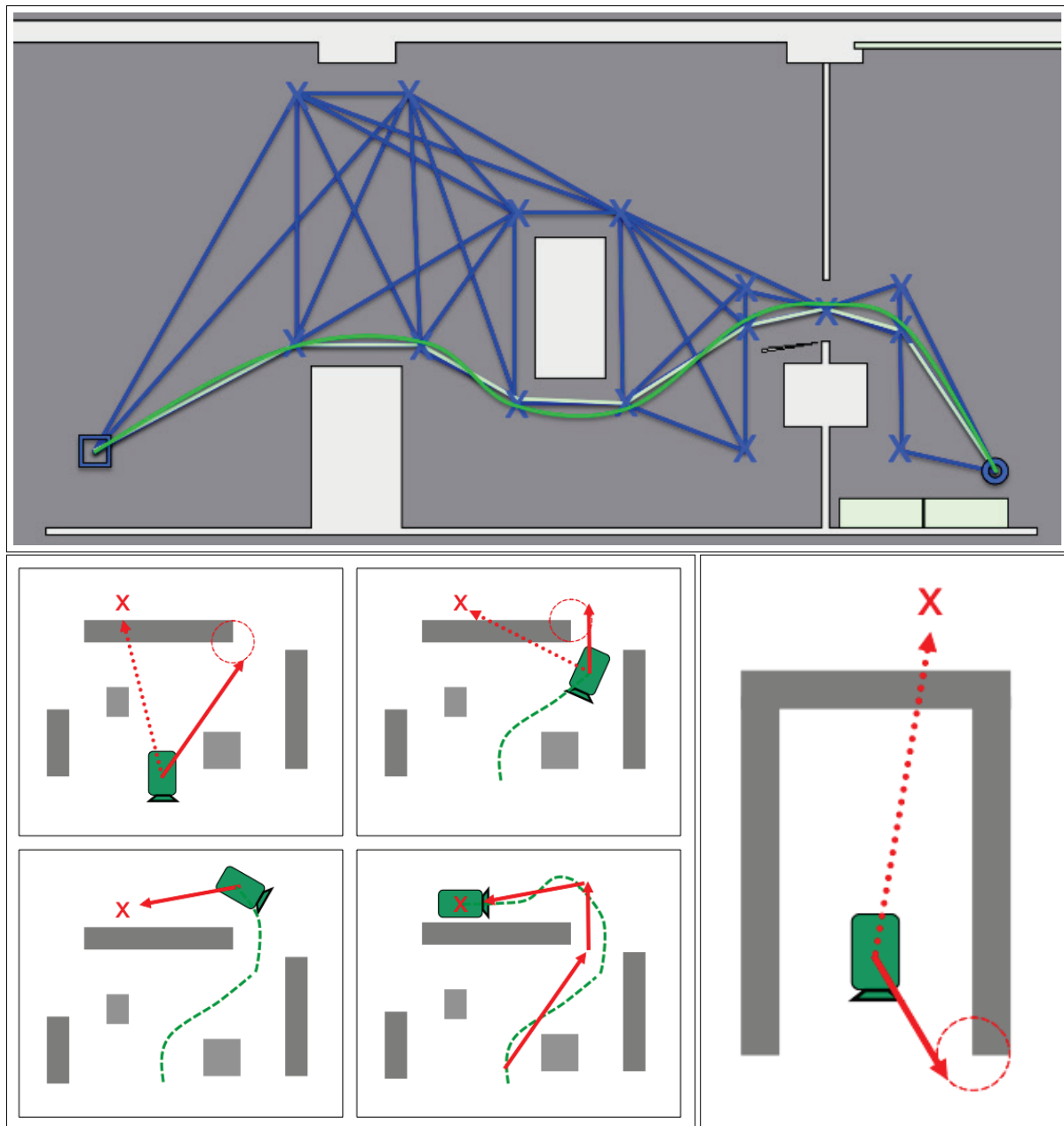


Fig. 4.3.: Predictive obstacle avoidance based on a geometrical analysis of the scene:  
 Top: Navigation according to the *tactical behaviors* of the motion model of pedestrians (dark blue edge-mesh along the local geometry). The mesh is calculated only locally one step ahead – **not** globally.  
 Left: A sequence of local sub-targets is generated at the obstacles' corners, forming the path to the target location (red X). The result is some kind of local dynamic visibility graph.  
 Right: The robot automatically avoids U-shaped obstacles or moves out of them by choosing the best suited corner as sub-target which is usually located at the exit.



#### 4.1.5. The *Hardware Abstraction Layer*

The *Hardware Abstraction Layer* is the lowest layer of the control architecture. From an behavioral or ethological point of view this layer provides the embodiment which is one of the characteristics and at the same time a necessary condition of *Behavior-Based Controls*: this layer connects the *BBC* to the real world. Information on the actual hardware setup of *InBOT* and *ETrolley* can be found in Annex B.

From an architecture point of view it is intended to make the remaining control system independent of the actual robotic platform and the used sensor systems. Thus, it consists of two parts: the *Platform Abstraction* and the *Sensor Abstraction*.

The *platform abstraction (PA)* receives a 3D velocity set-point vector  $(X, Y, \theta)$  from the upper control system in defined time intervals. Depending on the actually available drive system, this vector is either directly transformed into motor commands, such as in the *Mecanum* drive of *InBOT* which has full 3 DOF, or the dimension is reduced by adding a time constraint. This is necessary for example when using a castor drive or a differential drive. As information for the upper layers the PA provides a status variable and the odometry values of the platform. Using a differential drive, these can be acquired from the driven wheels itself, in the case of a *Mecanum* drive extra hardware is needed for precise values. Additionally, the PA controls the (de-) acceleration restriction which is based on the motor power and the friction of the wheels.

The *sensor abstraction (SA)* provides various information and stores them in the *local world model*. This includes for example an occupancy map of the local environment (Fig. 4.4 – left). In the current version it is acquired on all platforms by planar laser range finders but the same grid could be generated by 3D sensors or stereo vision by projecting the obstacles on the ground plane. The size of the local map is currently set to 100 cells edge length with a size of 10 cm for each cell. As the robot is centered in the map it has a distance of view of approximately 4.5 meters.

In addition to the environment, the forces measured by a force sensitive device are transformed to a 3D force vector  $(X, Y, \theta)$ . Information on the transition of topological areas are gathered and handed over to the control system to match them with the topological model of the environment. On *InBOT* and *ETrolley* these information are acquired by a short-range RFID reader and the ground mounted RFID-barriers, but various other possibilities like Bluetooth curtains or vision-based systems are also possible.

**Data pre-processing: segmentation of obstacles and *object database*** Based on the occupied cells of the map individual obstacles are segmented and stored in an object list (Fig. 4.4 – right). The module *GridToObstacles* of the data processing tool chain is responsible for identifying the static obstacles of the *object database* – basically a list of all local obstacles, accompanied by characteristic data like position of the start and end point, size, and in case of moving obstacles their velocity, the history of their positions, and so forth.

The module segments continuous obstacles out of the local occupancy map of the *local world model*. The map is scanned for occupied but not marked cells. If such a cell is found, it is marked and the cells belonging to the same obstacle are marked as well by using a flood fill algorithm. The algorithm is modified to take a larger neighborhood of each cell into account, to be able to skip over small gaps in the obstacle. This way, all obstacles are merged between which the robot can not pass through. Finally, the characteristic data of the segmented obstacle is extracted and added to the *object database*.

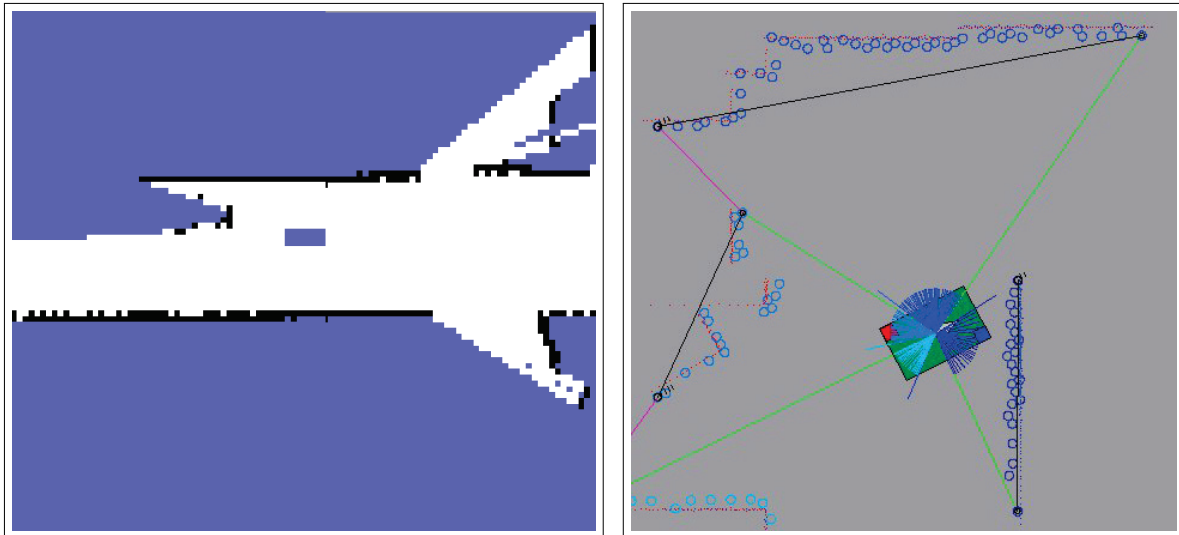


Fig. 4.4.: Left: Local occupancy map with a size of 100 x 100 cells. Each cell has a size of 10 x 10 cm.

Right: Extracted obstacles. The occupied cells of an occupancy map are segmented to define individual obstacles. Each individual obstacle consisting of interconnected cells is marked here in another shade of blue. The green lines indicate the starting cell of each obstacle and the black line connects the first with the last cell. Cells are counted as connected as long as the gap between them is smaller than the minimum size of the robot. Each of the tiny blue and red dots represent a single measurement of the laser range finders.

## 4.2. Summary of the self-localization concept

Before starting with the details on the *behavior-based* control system, this section summarizes the system implemented for the self-localization of the robot (see Chapter B.4 “Self-localization” and [Hei09] for a more detailed description). As defined by requirement R3 in the introduction, no detailed global map is available. Hence, popular scan-map-matching methods for the self-localization cannot be applied. Furthermore, it has to be expected that the robot is often surrounded by large numbers of people, impairing a continuous localization based on wall-mounted landmarks. Techniques would be needed which are independent from crowded corridors, colorful or low-contrast background and changing light conditions. Additionally, the landmarks would have to be sparsely distributed to keep the integration effort as low as possible.

### 4.2.1. Odometry for local self-localization

The local self-localization is responsible for tracking the local motion of the robot as precisely as possible. It uses two components: The first one uses incremental encoders to track the robot’s relative pose and position changes, the second one is an optional gyroscope to track the orientation changes more precisely.

- A gyroscope<sup>1</sup> is used on *InBOT* in order to track the orientation of the robot.
- A pair of 2D measurement wheels (see Fig. 4.5 and Appendix B for more details) was constructed to measure the linear – therefore the lateral and longitudinal – movements of the holonomic robot.

Following the idea of the University of Michigan Benchmark Test (UMBmark [23]) the self-localization of the system was tested. At the endpoint the average error was 0.8% with a variance of 0.00137. The maximum error was 1.4% of the driven distance (again more details can be found in Chapter B.4).

<sup>1</sup>LITEF micro-force6

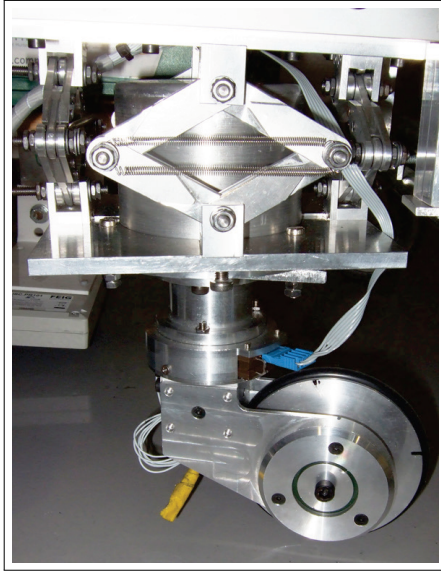


Fig. 4.5.: The 2D odometry wheel tracks the path its centerpoint has travelled. It measures the distance which it has rolled off and the direction of the rolling motion relative to the robot using wheel encoders. To smooth uneven floors it is mounted on a spring mechanism.

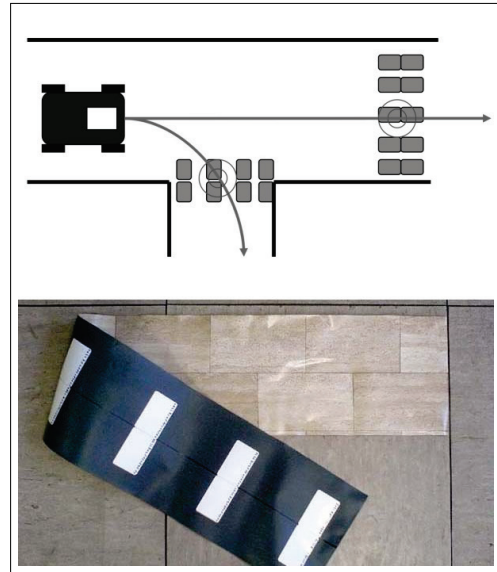


Fig. 4.6.: Top: Placement of RFID barriers to divide a corridor in different areas. Picture of an RFID barrier (bottom) with RFID transponders glued beneath a piece of PVC flooring.

#### 4.2.2. Global self-localization by floor-mounted RFID barriers

As commonly known, the accumulated position tracking has to be corrected due to also accumulating errors. Ordinary landmarks for global self-localization can be easily occluded by people or objects located in the line of sight. Wall-mounted long-range RFID-based systems can be blocked as well. To avoid these occlusions and to get more precise position information the presented concept proposes the use ground-mounted RFID transponders as landmarks as illustrated in Figure 4.6. This way an occlusion of the landmarks is hardly possible. Additional information and results can be found Chapter B.4, and in particular in the mid-study thesis by F. Steinhart [Ste08].

The relative position of the barrier, the position of the tags inside the barrier, and the time stamp when the first tag was read are known. The robot's position along the barrier can be estimated by averaging over the position of the read tags. The position orthogonal to the barrier is measured by comparing the time stamp when the first tag was detected and the last tag was not detected any more with the diameter of the coupling area of the reader and the current velocity.

### 4.3. The *behavior-based* control system implementing the tactical and reactive layers

After the fundamental information – namely drive system, sensors, maps, and self-localization – were briefly introduced, the remaining part of this chapter will in detail present the *Behavior-Based Control (BBC)* including the individual behaviors.

This section first describes the *BBC* itself, which was developed in the course of this thesis. As mentioned earlier, the organizational concept of the *BBC* strongly refers to the work by J. Albiez. Afterwards, the individual behaviors developed for the navigation system will be analyzed in a bottom-up order starting with

the safety behaviors and finishing with the geometric scene analysis. To enhance readability, the behaviors for the avoidance of moving obstacles and for multi-robot coordination will be presented in two dedicated chapters following the present one. Sticking to the concept of the human motion patterns, the individual behaviors developed here are – just as the overall control architecture itself – inspired by the movement behaviors of human beings, if applicable.

### 4.3.1. Architecture of the *Behavior-Based Control*

In the subsequent two paragraphs the concepts of the *Behavior-Based Control* of this thesis are described. The *BBC* is implemented by *Behavior Networks* which again consist of several individual behavior modules, or groups of behavior modules. Figure 4.7 illustrates how the individual behaviors are implemented by *MCA Modules* of the utilized framework *MCA2* (see Chapter 3.2.2 – “Fundamental concepts of the control architecture” for a short introduction).

**The behavior module:** The behavior modules, known from works by J. Albiez from the walking machine domain ([3]), were modified and adapted to the needs of the shopping cart scenario (Fig. 4.8). They are small software-modules that are dedicated to a single task. All behavior modules of such a *behavior-based* control system work independently and in parallel at all the time. They range from reactive like turning away from the wall to deliberative like freeing the way for another robot. Typically, the lowest level behavior modules, the reflexes, generate the motor commands while the higher behavior modules orchestrate the lower ones.

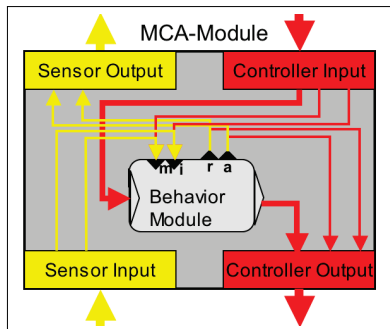


Fig. 4.7.: The behavior module is encapsulated and implemented by an *MCA Module*. The behavior’s interfaces are mapped to the MCA interfaces:  $e, i, m \rightarrow CI, SI$  and  $u, r, a \rightarrow CO, SE$

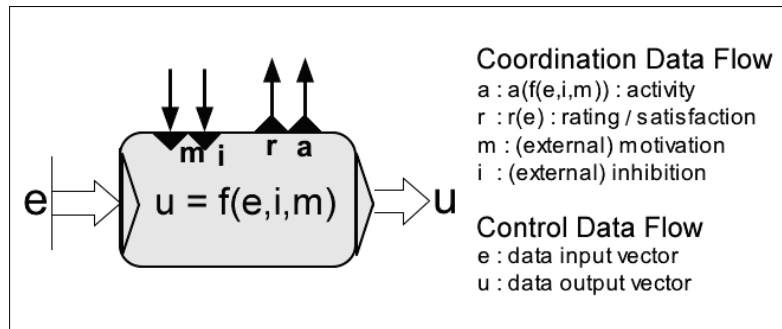


Fig. 4.8.: The behavior module is the basic unit of a *behavior-based* control system. It fulfils a dedicated task. The control data flow with the interface ( $e$ ) and ( $u$ ) is strictly discriminated from the coordination data flow with the interface rating ( $r$ ), motivation ( $m$ ), activity ( $a$ ) and inhibition ( $i$ ).

Besides calculating the dedicated output ( $u$ ) based on given input ( $e$ ), which represents the control data flow, behavior modules have the possibility to interact with each other directly, which represents the coordination data flow. A behavior can be motivated ( $m$ ) or inhibited ( $i$ ) by other behaviors, and each one can inform others about how active ( $a$ ) it is and how satisfied it is with the actual situation ( $r$ ). This information can be used by other behaviors to estimate how efficiently the first one is working and accordingly motivate or inhibit it.

The behavior modules used here accept an inhibition and a motivation value in contrast to Albiez’s “activation” value ( $t$ ). The two individual values offer more flexibility in fusing the coordination data flow,

especially when many modules are involved. The motivation ( $m$ ) states how much other behaviors want a special behavior to operate. Accordingly, it makes sense to fuse individual motivation values using their maximum. Otherwise, behaviors which are idle or don't have an opinion could down-rate a behavior. The opposite counts for the inhibition ( $i$ ). It states how much a behavior wants to suppress another one. Here again it makes sense to follow the strongest input. To get a defined design paradigm, in case of conflict the inhibition should overrule the motivation, meaning if one inhibition input equals zero the output ( $u$ ) has to be zero. This results in the following transfer function  $f()$  and functions for activity and rating, each for  $x$  inhibition and  $y$  motivation inputs:

$$u = f(e, i, m) = B(e)(1 - \max_x(i_x)) \max_y(m_y) \quad (4.1)$$

$$a = \|f(e, i, m)\| \quad (4.2)$$

$$r = \|e\| \quad (4.3)$$

$$(u, e) \in \mathbb{R}^n; \quad (i, m, a, r) \in \mathbb{R}, [0, \dots, 1] \quad (4.4)$$

**Behavior Networks:** Using these described capabilities, individual behavior modules can be woven into *Behavior Networks* as introduced by J. Albiez in [3]. In such a network a whole set of behaviors combines the individual abilities or functions to fulfill a higher task. The basic coordination of the *Behavior Network* is done in two ways: task-specific behaviors are activated from the outside by setting their motivation or the behaviors motivate and inhibit each other by connecting the signals  $a$  or  $r$  with  $i$  or  $m$  (see Fig. 4.9). Additionally, a special kind of very simple but powerful behavior module is used: the fusion behavior modules (Fig. 4.10). These allow the manipulation of the control data flow by unifying several data inputs of identical type to one data output of the corresponding type. This is necessary when several modules provide input for one and the same other module. The most prominent types are the maximum fusion, where on input is selected based on the activity of the source module and the weighted fusion where all inputs are weighted by the activity and then merged for the output. Figure 4.11 sketches the *Behavior Network* which implements the *tactical* and the *reactive layer*.

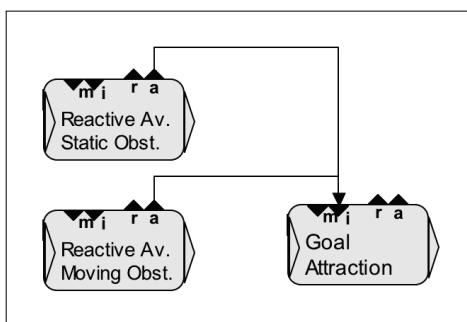


Fig. 4.9.: Coordination data flow: The two behaviors for reactive obstacle avoidance uses their activity output  $a$  to reduce the activity of the behavior “Goal Attraction” by setting its inhibition input  $i$ .

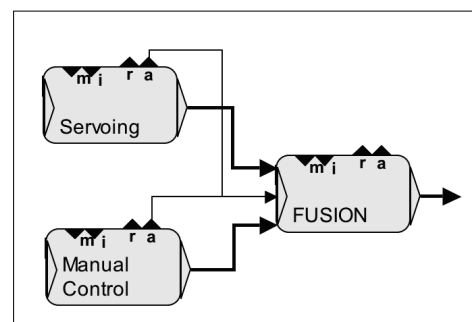


Fig. 4.10.: The control data flow: The output of two behaviors is merged by a fusion behavior. To decide which control data has priority the activity  $a$  of the two behaviors is provided.

Due to this loose coupling of the individual modules, the available functions can easily be recombined, activated or deactivated to fulfill a new specific task. They even can be motivated halfway only, to support

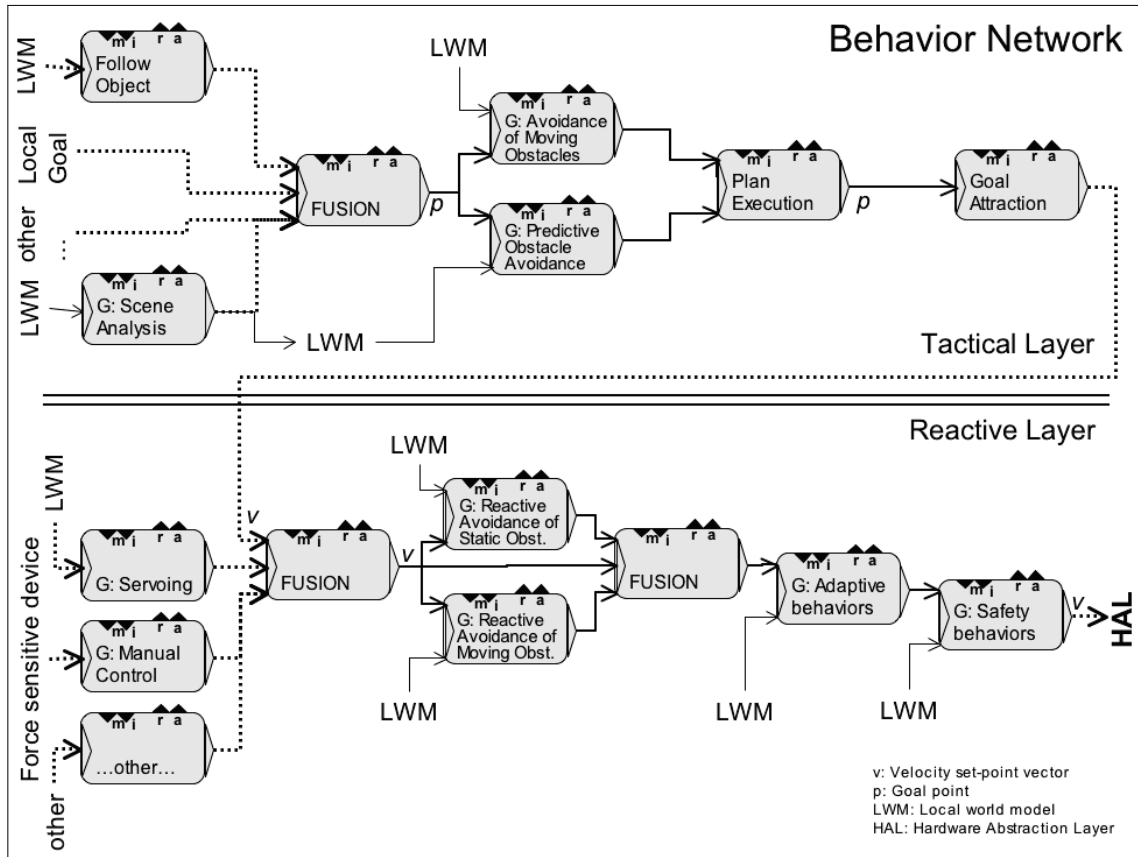


Fig. 4.11.: Abstract view on the *Behavior Network* showing the various behavior groups (indicated by “G:”) of both the *tactical layer* and the *reactive layer*. Each layer consists of four major steps: input, fusion of inputs, processing, and output. While in the *tactical layer* goal coordinates are the common language, the velocity set-point vectors play this role in the *reactive layer*. The behavior groups are only connected by the control data flow, the connections used for the coordination data flow ( $a, r, i, m$ ) are used inside the groups only. This way the influence of the behaviors is limited.

some other functionality. This way for example a slight and discreet obstacle avoidance augmentation for the *Manual Steering Mode* can be implemented without effort. Another advantage of the loose coupling and the powerful fusion behavior modules is the easy expandability of the network. New functionalities can simply be applied by hooking in their corresponding modules.

After introducing the architecture of the *BBC* in the subsequent section and paragraphs the individual behaviors will be described, starting with the safety behaviors.

#### 4.4. The reactive behaviors (RB)

This section elaborates on the behaviors which make up the *Behavior Network* of the *reactive layer* (see Fig. 4.11). The individual behaviors will be discussed in an bottom-up order. The most important behaviors are for safety, adaptation to the local scene, and obstacle avoidance. The “RB:” in the sections’ titles indicate the behaviors as being “reactive behaviors”.

#### 4.4.1. RB: Safety behavior

Ensuring safe motions is a crucial requirement for *service robots*. A lack of safety would result in hurt people, damaged objects, or at least in a lost of confidence and acceptance by the users. This requirement is tackled in this control system by applying dedicated safety behaviors. These are the last behaviors in the hierarchy and therefore the last ones which can manipulate the velocity set-point vector before the motor commands are generated.

The safety reflex is the lowest level behavior found in the presented control system. It restricts the maximum allowed velocity for the robot for each possible driving direction – discretized in one-degree steps to reduce the demand of computational power. Having a holonomic platform this task is more complex than on differential driven ones because the robot can freely accelerate in any direction without having to turn first.

For each individual range scan by the sensors (in the case of *InBOT* two SICK S300 laser range finders, scanning 360 degrees six times a second) the critical directions are calculated – therefore the two directions in which the body of the robot is just able to pass an obstacle on the left and on the right side. The maximum allowed velocity between these two directions is reduced.

The safety is mainly implemented by two modules: the module for data pre-processing “Scan to Safety” which generates a lookup table in the *local world model* and the reflex “Safety” which uses the lookup table to reduce the velocity of the robot.

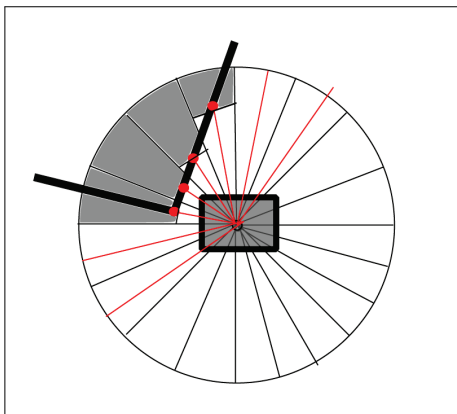


Fig. 4.12.: Sketch illustrating the sectors defined in polar coordinates. In each sector the sensor reading with shortest distance to the robot’s hull is stored. Beyond this value the complete sector is interpreted as occupied. The area between the robot and the value is interpreted as free.

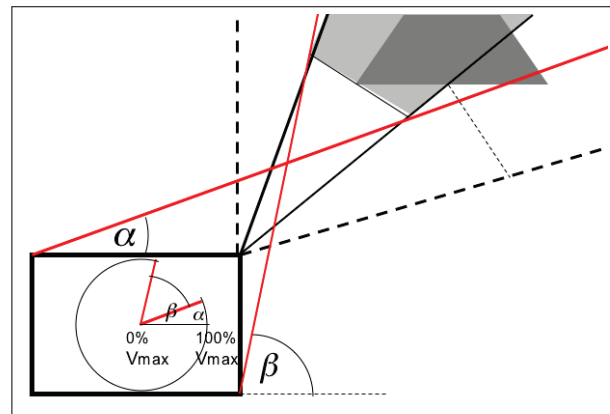


Fig. 4.13.: Sketch showing an obstacle and the corresponding critical directions (red lines). An obstacle (grey triangle) lies within an exemplary sector. The two critical directions  $\alpha$  and  $\beta$  are calculated, in which the robot is just able to pass without intersecting with the blocked part of the sector (red lines). Accordingly, the maximum allowed velocity is reduced between these two directions as illustrated by the circle in the robot’s middle.

**Scan to Safety – data pre-processing:** This is done by setting up a lookup table in polar coordinates, representing the 360 degrees around the robot in an discretized manner. The individual distance values of a scan (or readings from whatever other sensor used) are sorted into this lookup table, always storing the minimum distance for each sector. After processing a scan, each sector contains the shortest (actually measured) distance between robot and an object of this sector as illustrated in Figure 4.12. Afterwards, for

each occupied sector the two driving directions  $\alpha$  and  $\beta$  are calculated, in which the robot could drive just without intersecting with the sector beyond the occupied distance (see Fig. 4.13 for an illustration). These two directions are kept unconstrained, for the directions in between ( $v_{Sec_j}$ ), the maximum allowed velocity is limited according to the maximum possible de-acceleration of the robot, the current velocity, and the time delays:

$$v_{Sec_j} = \sqrt{2(d_O - d_d)a_{MaxDec} \cdot f_p} \quad (4.5)$$

$$d_d = v_c(t_d + t_p) \quad (4.6)$$

with  $d_O$  = distance to obstacle,  $d_d$  the distance traveled in the delay depending on the time between two laser scans ( $t_d$ ) plus the processing time ( $t_p$ ) and the current velocity ( $v_c$ ),  $a_{MaxDec}$  = Maximum possible deceleration and  $f_p$  a proportional safety factor.

The velocity restriction is again written into a lookup table which uses one degree angular steps by computing the minimum of the new and the already present value, if existing. The transition between differently constrained sectors is smoothed by averaging less constrained sectors with the adjacent stronger constraint ones (always rising the constraints). The value is accompanied by a time-stamp to be able to “forget” old values, hence to reset sectors or slots when no new updates appear. This lookup table is illustrated in Figure 4.14 where a blue line is drawn for each sector with the length corresponding to the maximum allowed velocity in the corresponding directions.

The same procedure is performed for the rotation velocity, here only for two individual directions: turning left and right.

**Safety Reflex** When the upper navigation system wants to steer the robot in a certain direction  $j$  with a certain velocity  $v_0$ , a corresponding velocity set-point vector is handed down to the safety reflex. For the desired driving direction the constraint is looked up in the *LWM* ( $v_{Sec_j}$ ) and the initial velocity set-point is reduced to a new velocity set-point ( $v_n$ ). The safety for the rotation is calculated likewise with angular velocity, angular deceleration and angular difference.

$$v_n = \min(v_0, v_{Sec_j}) \quad (4.7)$$

**Multiple instantiation** To be more robust versus illegal memory operations of other modules or similar critical errors – which cannot be totally ruled out in such an in-homogeneous system – the safety module is instantiated three times. These are executed in a sequential order to tackle timing conditions. A module “PassTrough” precedes the three safety modules (see Fig. 4.15). It passes the needed information to the tree instances and ensures that all three instances work based on exactly the same data. Additionally, it houses a watchdog to make sure the upper part of the control system is still alive. A module “MajorityDecider” succeeding the three instances checks if all three instances provide the same velocity set-point as output. If this is the case the new set-point is passed to the drive system. If the set-points differ a critical error occurred somewhere and the robot is stopped immediately. The critical component is the data preprocessing as it is only executed once due to the higher computational effort. But as minimum functions are used on multiple sensor readings, the impact of external influences is less hazardous. To rise safety further the models would



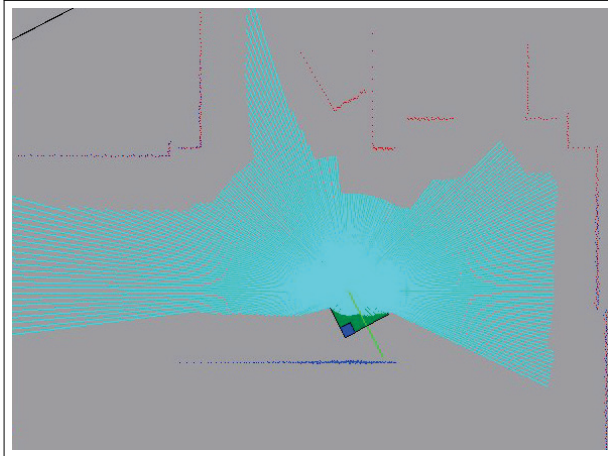


Fig. 4.14.: Screenshot showing the maximum allowed velocities in one-degree steps: to the left the velocities are unconstrained. At the bottom an obstacle is detected (blue dots representing the laser range finder measurements) so the robot is hardly allowed to drive in this direction.

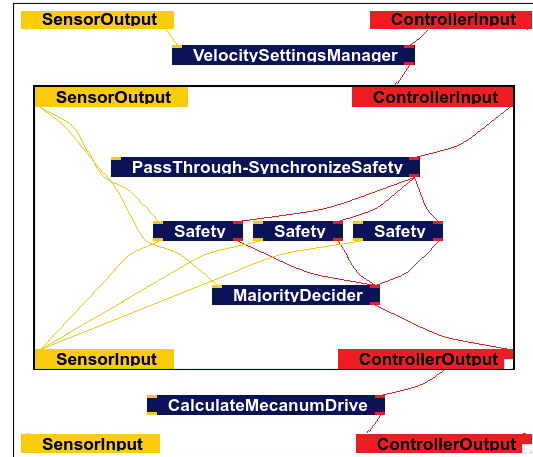


Fig. 4.15.: MCA-Browser screenshots showing the architecture of the safety group: The safety group receives a velocity set-point vector from the “VelocitySettingsManager”. The module “PassThrough” passes this set point on and ensures that all three instances of the “Safety” module receive the same velocity set-point. The “MajorityDecider” module collects the modified velocity set-points and checks if all three have the same value.

have to be implemented in three different way instead of just using three instances of the same module. Also running them on different computational units would bring major benefits.

#### 4.4.2. RB: Behaviors for adapting the velocity to the task and the user

As input this group of behaviors receives the velocity set-point vector from the fusion stage below the *Avoid Obstacle* behaviors. As shown in Figure 4.11, this group is the last but one group in the hierarchy. The group is responsible for reducing the the length – not the direction – of the velocity vector according to the individual field of duty of the behaviors. If no adaptation is necessary, they just hand the given vector down to the safety behaviors. The group is made up by three behaviors:

1. Guide User: This behavior slows the robot down when guiding the user, thus operating in the *Guiding Mode*. It implements a part of the *control sharing*. As additional input the behavior receives the distance to the user. If the user falls back, the behavior reduces the length of the velocity vector accordingly. The information on the user was acquired by two alternative systems: the *user tracking with onboard sensors* (see Appendix C.2 “Detecting and tracking the user using onboard sensors” and [63], [59]) as well as the *intelligent environment* (see Appendix C.3 “The intelligent environment”).
2. Adjust Velocity: This behavior is controlled by signals from the *strategic* or *communication layer*. It is responsible for slowing down the robot when the highest components in the control hierarchy wish. This usually happens when communicating to the user: the robot shall give the user time to react, not stressing him. The behavior accepts either a time-span or coordinates together with a distance as control input. This functionality has for example been used for studies with TU-Vienna and KTH Stockholm where the robot slows down to give the user a hint on products which are currently passed (for example [78]). Here the command was sent by the *communication layer*, implemented by the

*CR-UI* (see [87] and Appendix C.5). Another application is the advertisement of products like done by the second alternative for the *communication layer*, the *InBOT-UI* (see Appendix C.4).

3. Velocity Settings Manager: This behavior adapts the velocity set-point vector given by the upper part of the control system to general restrictions posed by the system's configuration, the current *mode of operation* or other sources. It ensures that the robot does not (de-)accelerate too fast regarding the friction of the wheels and even more softly when operating with a user in the close vicinity.

#### 4.4.3. RB: Avoid Obstacles (AO) – behaviors for the reactive obstacle avoidance of static obstacles

This section presents the behaviors responsible for the reactive avoidance of static obstacles. They have an important role in the execution of the plan which was generated by the *global navigation* in the first place and then refined by the *tactical behaviors*. The output of the AO behaviors are merged with the output of other behaviors, such as the target attracting one, by a fusion behavior. The result is handed down to the adaptive behaviors described in the last section and finally to the safety behaviors. This group is the last one in the hierarchy which manipulates the direction as well as the length of the velocity vector (see Fig. 4.11). The lower groups are allowed only to reduce the length.

The AO behaviors are working based on the occupancy map from the *local world model* and therefore have to consider a large number of individual cells. To tackle this, the idea is to merge all relevant obstacles in the local environment to individual representative points by making some kind of a weighted sum of the occupied cells – the weight depending on the cells' relevance or danger it poses to the robot. In the next step, all representative points contribute a repelling velocity set-point vector. These are finally merged with the target attracting vectors to provide the final velocity set-point vector.

Two versions of the AO behavior modules were implemented. The first one is straight forward: it uses repelling vectors pointing from the obstacles towards the robot's center. But this concept proved to slow the robot down too much. Therefore the second version uses repelling vectors which are orthogonal to the robot's driving direction, mainly altering the robot's course, not the velocity. Both versions are introduced following after the next paragraph which describes the calculation of the grid cells' relevance which is important for both versions.

**Relevant obstacles and relevant directions:** The reactive behavior modules responsible for the avoidance of obstacles are – according to the data abstraction hierarchy – working on basic data only: the occupancy grid map. As the grid contains a large number of cells, it is crucial for the behaviors to focus on the most important cells and not to waste effort on cells which will hardly effect the robot. Of high importance are such cells which lie within the driving direction of the robot and which are close to the robot. But depending on the used drive system even such a simple rule can become complicated (see sketches in Fig. 4.16). When the robot is using a holonomic drive like a *Mecanum* drive instead of a differential one, there are two direction to be considered: the one the robot is currently driving in as well as the direction in which it would want to accelerate in. Usually this second direction is the direction of the goal, but it can result from the input of other sources as well like the *force sensitive handle*.

To solve the situation for the general case it is defined that one module is only responsible for one relevant direction. If more than one direction is relevant, the module has to be instantiated twice, or even more often.

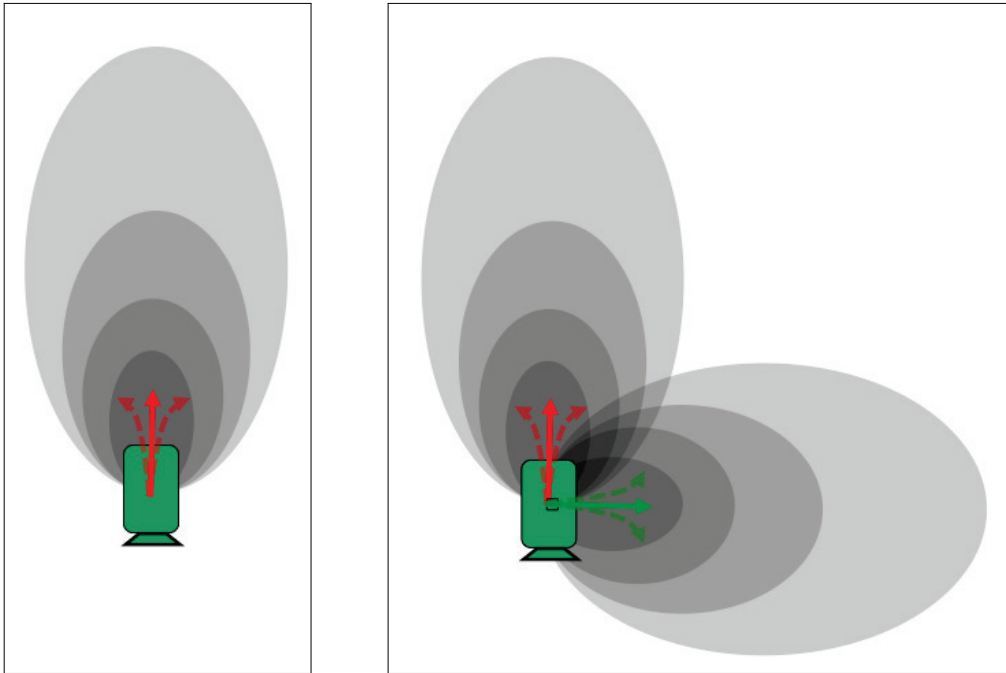


Fig. 4.16.: Sketch showing the *area of attention (AoA)* of the robot, hence the relevance of obstacles regarding the robot's position and heading: the darker the area, the more important are obstacles lying within it. Indicated with red is the current driving direction, indicated in green is the direction of the target, therefore the direction in which the robot wants to accelerate.

Left: the robot uses a differential, castor, or similar drive.

Right: the robot uses an holonomic drive.

To limit the number of cells a module has to take into account, an *area of attention (AoA)* is defined. All cells outside have an importance value of zero. Those cells inside are weighted by the distance to the robot and the angular distance compared to the corresponding relevant direction.

### RB: *Avoid Obstacles – Version 1*

As introduced, these behavior modules are working based on the occupancy map and therefore have to consider a large number of individual cells. The idea is to merge all relevant obstacles in the local environment to individual representative points by making some kind of a weighted sum of the occupied cells – the weight depends on the cells' position compared to the robot's position, the robot's velocity and intended driving direction. After this, all representative points contribute a repelling vector: the vector points from the representative point to the robot's center and has a length corresponding to the weight function. In the first version of the behavior modules, one instance is created for each, the left and the right hand side. This way two repelling vectors are generated, one for each side. Thus the robot is able to move through narrow gaps – if only one representative point would be created, this would be located right in the middle of a gap, blocking the robot just as if the robot was approaching a wall. Figure 4.17 illustrates the concept: left a photo of the scene is displayed and on the right the occupancy map containing the robot and the vectors is sketched.

**Relevance of grid cells:** The cells of the occupancy grid ( $\vec{C}_{i,j}$ ) inside the *area of attention (AoA)* are weighted with the function  $W_{rel}()$  by their distance to the robot ( $d_{rob}()$ ), their angular difference ( $\theta_{D_{rel}}()$ )

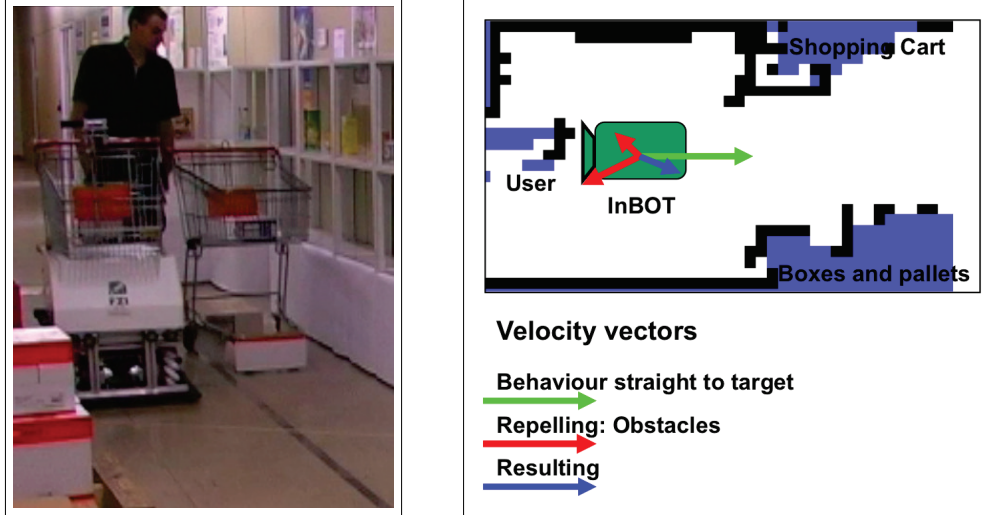


Fig. 4.17.: A scene illustrating the function of the *Avoid Obstacle* behaviors. The robot (here an early version of *InBOT*) is guiding a user and passing through some obstacles. The repelling vectors (red) are merged with the target attracting vector (green) to a final velocity set-point vector (blue) – which significantly smaller than the original green vector.

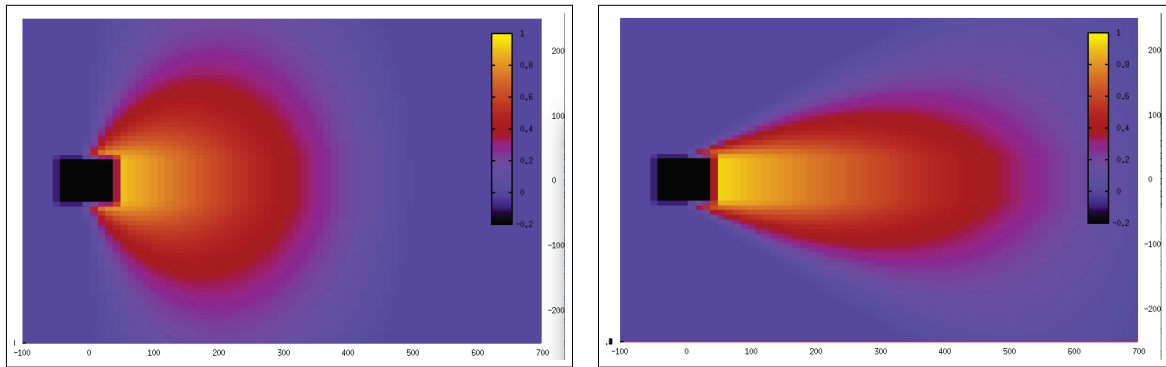


Fig. 4.18.: Plots showing the relevance of the cells of the occupancy map in the X-Y plane. The black rectangle is the robot, the relevant direction is horizontally to the right. The warmer the color the more relevant is the cell, in a range from 0 to 1. Parameters of the AoA left: of  $5m$  distance and  $\pm 90$  degree for fast turning vehicles; right:  $7m$  and  $30$  degree for fast moving and slow turning vehicles.

compared to relevant direction ( $\vec{D}_{rel}$ ) and their occupancy value ( $O(\vec{C}_i)$ ). Obstacles outside the *area of attention* (*AoA*) (limited by  $d_A$  and  $\theta_A$ ) are not taken into account. This reduces the influence of obstacles that are not in the (intended) driving direction, causing less disturbances. For example driving along walls or through narrow corridors becomes easier and oscillations can be avoided. Cells which are surely outside the *AoA* are omitted altogether, saving precious computational power. The relevance of the grid's cells is plotted in Figure 4.18 according to the following functions:

$$W_{rel}(\vec{C}_{i,j}) = \frac{\max(d_A - d_{rob}(\vec{C}_{i,j}); 0)}{d_A} \cdot \frac{\max(\theta_A - \theta_{D_{rel}}(\vec{C}_{i,j}); 0)}{\theta_A} \cdot O(\vec{C}_{i,j}) \quad (4.8)$$

$$O(\vec{C}_{i,j}) = 1 \text{ if } \vec{C}_{i,j} \text{ occupied, } = 0 \text{ if not occupied} \quad (4.9)$$

**Obstacles' relevance – the *center of disturbance* (CoD):** The position of the obstacles' CoD ( $\vec{P}_{CoD}$ ) is the *center of gravity* (GoG) of the obstacle, moved according to the relevance of the contained cells. Accordingly, all positions of the cells ( $\vec{P}_{C_{i,j}} = \begin{pmatrix} i \\ j \end{pmatrix}$ ) are weighted and then summed up to generate the CoD. To respect the fact that two instances of the behavior modules shall be used giving two CoDs, one each for the right and left side, the actual AoA is split along the relevant direction, giving two sub-AoAs.

$$\vec{P}_{CoD} = \frac{\sum_{i,j \in AoA} \vec{P}_{C_{i,j}} \cdot W_{rel}(\vec{C}_{i,j})}{\sum_{i,j \in AoA} W_{rel}(\vec{C}_{i,j})} \quad (4.10)$$

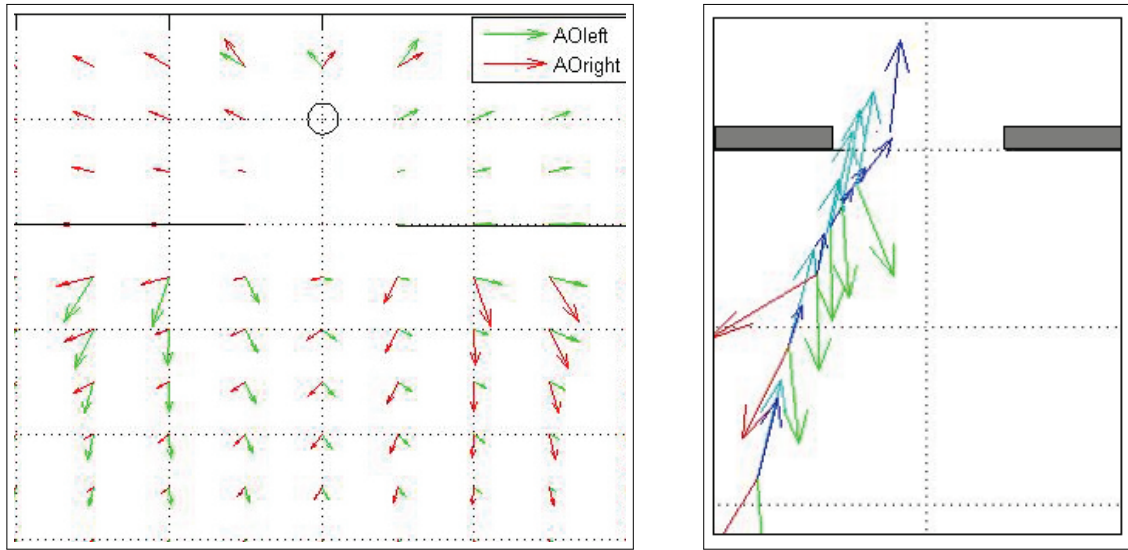


Fig. 4.19.: Matlab simulation of two instances of the *Avoid Obstacle* behavior when passing a door. It shows two repelling vectors (green and red for left and right hand side obstacles), the target's attracting vector in light blue and the resulting velocity set-point vector in dark blue.

Left: the pair of repelling vectors is calculated for several exemplary positions.

Right: the vectors are calculated for a series of steps in time.

**Output data – the vectors  $u$ ,  $a$ , and  $r$ :** The modules' repelling output vector  $\vec{u}_{AO}$  (part of the control data flow) is calculated in three steps. Before describing the individual steps, it should be mentioned that all calculations are performed in robot-coordinates, meaning the the robot's center point has always the coordinates ( $X = 0, Y = 0$ ) and the corresponding relevant direction the direction  $\theta = 0$ .

In the first step, the direction of  $\vec{u}_{AO}$  is determined as the normalized vector  $\hat{P}_{C-R}$  from the obstacles' CoD towards the robots center. Then the length of the vector is calculated by using the weight of the CoD itself and multiplying it with the robot's current velocity  $|\vec{v}|$ . This enables the robot to slowly approach an object but generates a strong deceleration if the robot approaches fast. Finally, the vector is adapted to the parameters for the external coordination, thus the input from the coordination data flow: the motivation ( $m$ ), and the inhibition ( $i$ ) as well as a proportional factor ( $f_p$ ) for the overall adjustment. Finally the two output values for the coordination data flow are calculated. The activity  $a$  of the behavior modules is the length of the repelling vector. The rating  $r$  is the total sum of the weights of the cells of the AoA and represents the

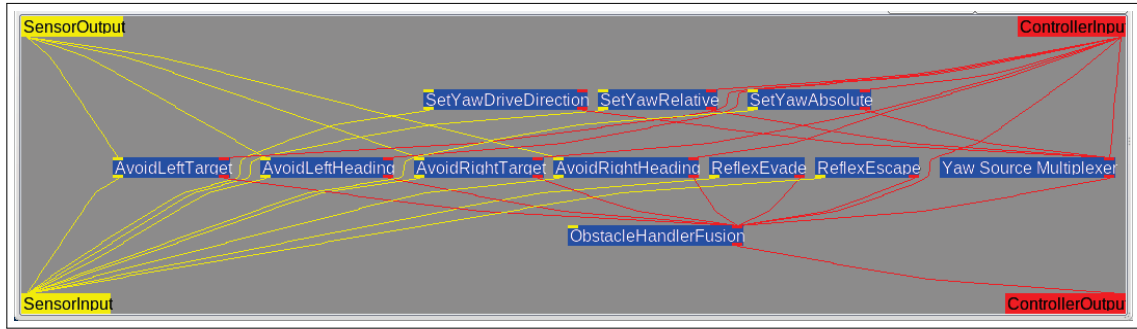


Fig. 4.20.: The implementation of the first version *Avoid Obstacle* behaviors in MCA: four instances were created (each left and right for the heading as well as the target direction) which are merged using a fusion behavior.

difficulty of the local environment regarding the amount and the relevance of the local obstacles – thus the “stress level” of the behavior.

$$\vec{u}_{AO} = \hat{P}_{C-R} \cdot W_{rel}(\vec{P}_{CoD}) \cdot \left| \frac{\vec{v}}{v_{max}} \right| \cdot f_p \cdot m \cdot (1 - i) \quad (4.11)$$

$$a = \|\vec{u}_{AO}\| \quad (4.12)$$

$$r = \sum_{i,j \in AoA} W_{rel}(\vec{C}_{i,j}) \quad (4.13)$$

The concept was validated using a Matlab simulation. Figure 4.19 for example shows vectors which have been calculated for a set of situations when approaching and then passing through a door. It can be clearly seen that the large repelling vectors from both sides of the door slow the robot down significantly.

**Implementation in MCA:** Figure 4.20 shows a screenshot from the MCA-Browser which displays the four instances of the *Avoid Obstacle* behavior modules whose velocity set-point vectors are merged with the vectors from other modules using a fusion behavior module.

### RB: *Avoid Obstacles – Version 2*

The first version of the behaviors for obstacles avoidance performed well when looking at the number of successful maneuvers. But the repulsive vectors slowed the robot down too much. When looking at human movement behavior, one can see that humans obviously bypass obstacles without slowing down much, just altering their course. Accordingly, the second version of these behaviors tries to generate repulsive virtual force vectors, which are orthogonal to the driving direction. In the fusion step, they will accelerate the robot sideways instead of slowing it down. A holonomic robot will drive diagonally while a robot using a differential drive will start to turn.

In addition to the direction of the repulsive vectors some other adaptations of the behavior modules were implemented:

1. **Only one module for left and right hand obstacles:** As the direction of the repulsive vector shall be orthogonal to the driving direction, the vectors for the modules responsible for the left hand and the right hand side are directly opposite and cancel each other out. This facilitates implementing both behavior functions in one module, easing the fusion and simplifying the network. Moreover,

both functions are always be motivated or inhibited with the same value, resulting in always balanced behaving.

2. **Three instances:** As described in the previous paragraph, the functions for left and right hand obstacles are combined in one module. This means that there are only two instances of the behavior module: one for the current driving direction and one for the direction towards the target. A third instance is introduced corresponding to the direction of a safety velocity restriction enforced by the safety behaviors. This ensures that the robot is repelled from the direction in which it was driving when it was stopped. Thus, the third instance ensures that the robot does not get stuck when a safety behavior stops the motion.
3. **Shape of the AoA:** The *area of attention* is bordered by a polygon and all calculations are done versus the individual lines of the polygon. This way the calculation of angular displacements for cells given in Cartesian coordinates can be omitted. As the symmetry axis of the AoA is always identical with the positive X-axis ( $\Theta = 0$ ), and the borders of the AoA are parallel to the axes, the weight function  $W_{rel2}$  is cheap to compute.

The behavior module is instantiated for all three relevant directions of the robot, each contributes one individual velocity set point vector:

**Avoid Obstacle behavior Target (AOBT)** for the intended movement direction corresponding to a given target or a velocity set point vector given by other means (*force sensitive handle*, joystick, etc)

**Avoid Obstacle behavior Heading (AOBH)** for the current heading or the current movement direction respectively

**Avoid Obstacle behavior Safety (AOBS):** AOBS predicts the direction in which the safety reflex further down in the control hierarchy is apt to slow down/stop the robot and uses this direction as direction of interest (but with a much smaller area of interest). The goal is to avoid being stuck due to the safety reflex.

**Output data – the vectors  $u$ ,  $a$ , and  $r$ :** To reduce the speed reduction caused by the behaviors, the repelling vector  $\vec{u}_{AO2}$  shall be as far as possible orthogonal to the relevant direction. As the function is calculated in the coordinate system where the relevant direction is equal to the X-axis, the repelling vector shall have a strong Y-component and a weak X-component. The weight function has proven well, so the weight will be basically kept, and so will be the length of the repelling vector. But in contrast to the first version where the direction of the vector was from the CoD to the robot, this time the direction will be parallel to the Y-axis (see Fig. 4.21 for a sketch). As the repelling vectors of AO-Left and AO-Right are exactly opposed to each other (Y and -Y direction), both are handled in one module, resulting in the repelling force  $F_{rep}$  which is the difference of the length of both vectors.  $F_{rep}$  provides the Y-component of the final repelling vector  $\vec{u}_{AO2}$ . The X-component is a decelerating force  $F_{dec}$  proportional to the strength of the

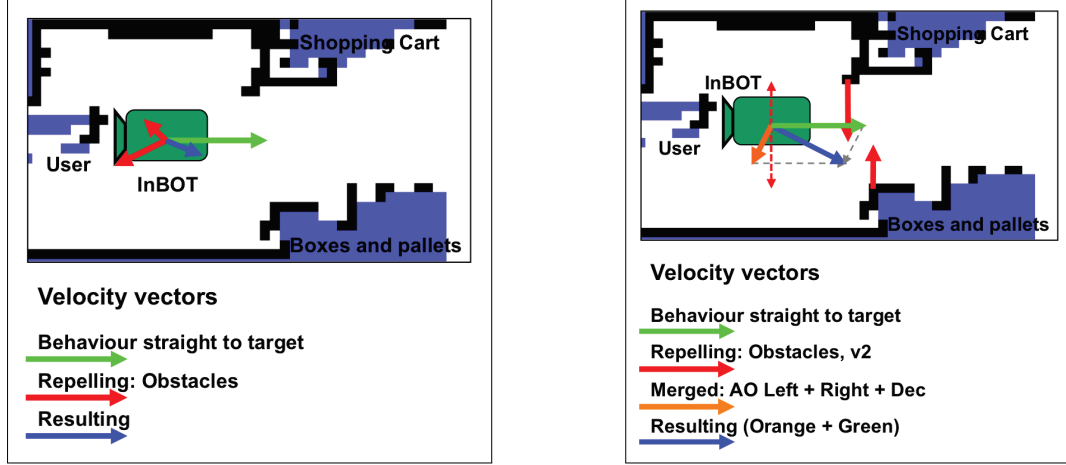


Fig. 4.21.: Comparison of the *Avoid Obstacle* behaviors (AOBH instance). Left: first version, right: second version. The red arrows indicate the repelling vectors from the left and right hand side functions – this time orthogonal to the relevant direction (here: direction to the target indicated by the green arrow). The dotted red arrows are the same ones, projected to the robot's center. By summing these two up, and adding an extra deceleration component we get the orange arrow. The orange and the green arrow are input to the fusion behavior which generates the blue resulting arrow. The resulting arrow – equal to the final velocity set-point vector – of the second version is significantly longer than from the old version.

individual repelling forces with an proportional factor with an value of about  $f_{dec} = 0.2$ . This changes the functions as follows:

$$F_{rep} = |W_{rel2}(\vec{P}_{CoD-left})| - |W_{rel2}(\vec{P}_{CoD-right})| \quad (4.14)$$

$$F_{dec} = \max(|W_{rel2}(\vec{P}_{CoD-left})|; |W_{rel2}(\vec{P}_{CoD-right})|) \cdot f_{dec} \quad (4.15)$$

$$\vec{u}_{AO2} = \begin{pmatrix} -1 \cdot F_{dec} \\ F_{rep} \end{pmatrix} \cdot \frac{|\vec{v}|}{|\vec{v}_{max}|} \cdot f_p \cdot m \cdot (1 - i) \quad (4.16)$$

$$a = |\vec{u}_{AO2}| \quad (4.17)$$

$$r = \sum_{i,j \in AoA}^{i,j} W_{rel2}(\vec{C}_{i,j}) \quad (4.18)$$

At first glance, merging the left and the right hand side CoD to one resulting CoD would ease the computation, but in this case the robot would not be able to pass doors, as the resulting CoD would lie in the middle of the free space. Using two CoDs, the two repelling vectors will cancel each other out when the robot is in the middle of the door, letting the robot pass with only slightly reducing the velocity.

An interesting situations is encountered when the robot drives exactly straight ahead towards a wall. Here the left and the right hand side vectors cancel each other out almost completely. When the target is exactly straight ahead and the wall is perfectly orthogonal, both behaviors AOBH and AOBT would not initiate a change of the course, just slowing down the robot. But this is an purely theoretical situation. In realty the following interaction takes place: due to some slight fuzziness (discretization, sensor precision, friction, uneven floor, and so forth) the actual driving direction varies a little bit. Thus the AOBH behavior starts to change the course in favor of one side. Only a little at first, but once the course has been changed, the favor for this side rises fast. To cover the theoretical possibility of a tie, the behavior *Break Tie* is introduced later, but is has never been active.



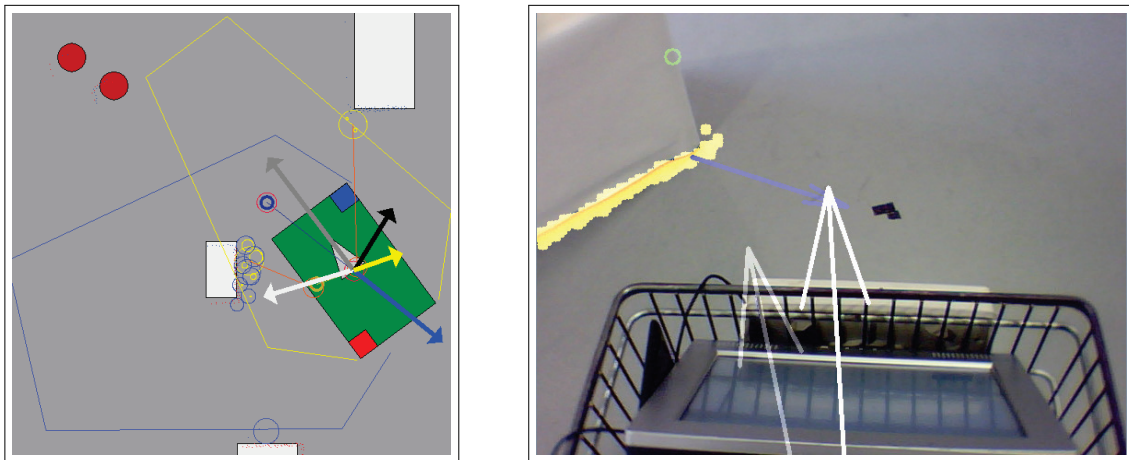


Fig. 4.22.: Left: Screenshot of the MCAGUI showing the behaviors AO – version 2 – in action. White arrow: current velocity vector, grey arrow: target direction, blue arrow: repelling vector AOBH, yellow arrow: repelling vector AOBT, black arrow: final velocity set-point vector – it counters the current velocity vector to prevent the robot from colliding with the left obstacle and slightly accelerates in the target direction. The blue and the yellow frames represent the areas of attention for obstacles, the small circles are occupied cells in the occupancy grid (the size of the circle corresponds to the relevance of the cell) and the large circles are resulting representative obstacle points. Right: Video snapshot showing the robot's perspective of a similar scene. The green circle is the goal to reach (grey arrow), but the path is obstructed by a shelf (yellow dots) so a repelling vector is generated (blue arrow). Finally all vectors are merged to a resulting velocity set pint vector (white arrow).

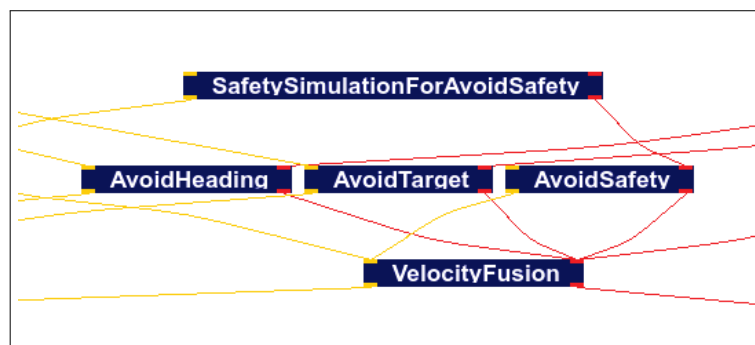


Fig. 4.23.: MCA-Browser screenshot showing the implementation of the second version *Avoid Obstacle* behaviors in MCA: here three instances were created (heading , target, and safety) which are merged using a fusion behavior. The fusion behavior module receives input from other reactive behaviors as well which can not be seen here.

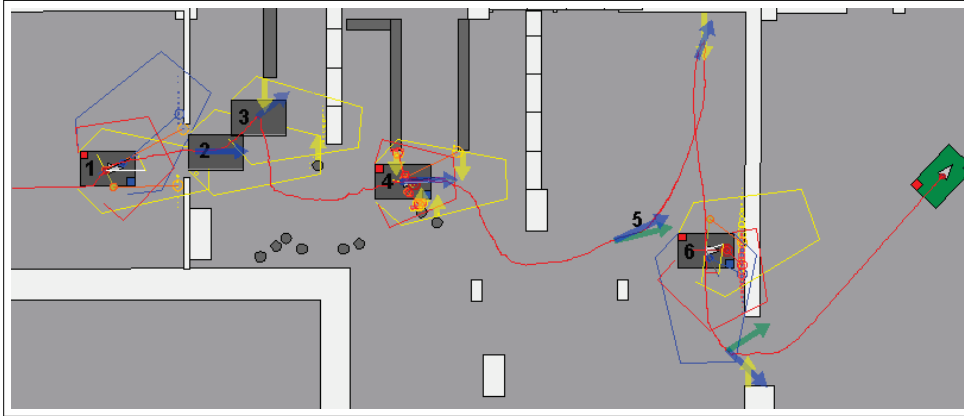


Fig. 4.24.: Annotated path generated by the *Avoid Obstacle* behaviors – version 2: starting on the left side the robot was ordered to move to its goal pose (green robot shape on the right side). The colored coffin-shapes indicate the corresponding areas of attention: the AoA of AOBT is marked in yellow, of AOBH in blue, and of AOBS in red. The green arrow points towards the target and the blue one is the resulting velocity vector. It shall be noted that the shown capabilities depend only on the current sensor information and on a given direction to the target. If this target direction is provided by a target finding behavior (as in this example) or by the *force sensitive handle*, a joystick, or by teleportation is not relevant. As long as a target direction is given, the *reactive behaviors* are operational and work on finding a safe path in the given direction. Six special situations will be commented here:

- 1) The robot approaches a door: the *Avoid Obstacle* behavior (AOB) regarding the target direction (AOBT-yellow) pushes the robot up while the AOB regarding the current heading of the robot (AOBH – blue) pushes the robot down. This way the robot is forced in the middle of the door.
- 2) Just after passing the door the robot encounters an obstacle on the front-right side, pushing the robot upwards. This is not a suitable path but due to the limited *field of view* (FOV) of the AOBs the robot cannot decide better. Usually the predictive obstacle handling of the *scene analysis* takes care of this.
- 3) As soon as the next obstacle enters the FOV of AOBT on the upper side, the robot detects the mistake made in 2) and turns downward.
- 4) Various obstacles in the FOV of AOBT force the robot in the middle of the passage.
- 5) Shortly after passing the white box, the right side wall enters the FOV of AOBT. AOBT does not know which way to move around the large obstacle so it favors the current direction. The robot keeps moving roughly in the same direction while adapting to the wall until either the angular error compared to the target direction becomes too large or until another obstacle is encountered. Usually this would be a situation to be solved by the predictive obstacle handler.
- 6) After turning around, the robot follows the wall in the other direction: the AOBT, AOBH, and AOBS (orange shape) and the target attracting behavior are in an equilibrium state which allows the robot to follow the wall without the need of an explicit wall following behavior.

**Incorporation of behavior modules and implementation in MCA:** Just like in the first version, here again all three repelling vectors are merged with the attracting vectors by a weighted fusion behavior module to generate the final velocity set-point vector. This is illustrated in the MCAGUI screenshot in Figure 4.22 on the left and from the robot's point of view drawn into the picture of an on-board camera in Figure 4.22 on the right hand side. Figure 4.23 shows a screenshot from the MCA-Browser which displays the two instances of the *Avoid Obstacle* behavior modules whose velocity set-point vectors are merged with the vectors from other modules using a fusion behavior module.

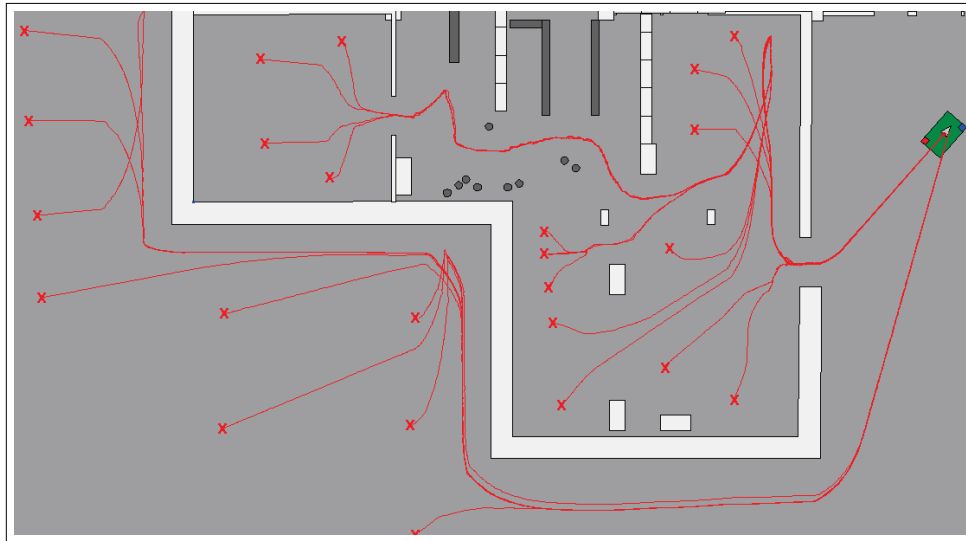


Fig. 4.25.: Validation: 24 test runs of AO-v2 (in simulation due to limited space in the labs) are shown in this figure. The robot was placed on the positions marked with the red “X”s distributed over the place. Then the robot was ordered to move to a certain target position (green robot shape). In all cases the simulated robot was able to reach the target position using the reactive navigation behaviors only – and only relying on current (simulated) sensor readings. In some cases the path was not efficient – this happens when the robot has to decide which way to move around an obstacle (compare to preceding figure). But taking care of situations as these is the responsibility of the predictive obstacle handling of the *tactical behaviors* discussed later in this chapter. No explicit wall following behavior was used – the wall following results from an equilibrium state of the target finding and obstacle avoidance behaviors.

## Evaluation

This paragraph provides test results for the second version of the *Avoid Obstacle* behaviors. Additional tests with the complete navigation system are presented at the end of this chapter and even more will be implicitly given in the subsequent chapters as the AO behaviors represent the baseline of the navigation system.

Figure 4.24 illustrates details on the behaviors’ functions during an exemplary path through a door and around some obstacles. The figure shows the interplay of the behaviors as well as the resulting explicit and implicit effects. The expected drawbacks of the limited field of view of the behaviors is visible as well – which will be countered by the predictive obstacle handling behavior of the *tactical behaviors*.

Finally 4.25 shows a larger scale validation of the concept. This test has obviously been performed in simulation because the necessary space is not available in the labs. The basis for the validation is a map of the complete lab including offices and outside areas and a detailed simulation of the robot platform down to each individual scan point of the laser scanners. Additionally, the simulation – for which large pieces could be gratefully taken from other work – incorporates the (de-) acceleration capabilities, the drive system and the S300 laser scanners including the corresponding precisions and variances. The simulated robot was placed on 24 different positions inside and outside the lab (a major advantage of the simulation: the outdoor space could be used as well) and commanded to move to a certain target location. Only the behaviors for the reactive avoidance of obstacles (*Avoid Obstacle* and safety behaviors) were used, operating on current sensor information only – no pre-knowledge has been available besides the starting and the goal coordinates. As the figure shows, the robot was able to reach the goal even when having to pass cluttered and narrow regions. The robot was able to cope with narrow spaces as well as with wider areas. Here again the effects of the limited field of view can be seen, resulting in temporary movements into wrong directions when

for example passages have not been in the area of attention. This test run also illustrates the deterministic behavior of *Avoid Obstacles*: the robot is always attracted to the same path, even when starting at different locations.

**Comparison to VFF and VFH** As the concepts presented here and the *VFF* methods (see Chap. 2.3.3, Fig. 2.13) both use virtual force vectors they look very similar, at first glance. But there are some major differences: In this thesis's concept, only one single representative point contributes a force vector, originating from the most dangerous location. Thus, the size of the obstacle or its density is not relevant. The *AO* behaviors are able to pass through gaps which are radial to the robot as well as through narrow gaps: in v1 the repelling vectors' strength is reduced when the robot slows down in front of the gap, in v2 the forces hardly push the robot back at all. The basic concept of the *VFH* – the polar histogram – is used only for reducing the velocity by the safety behaviors. Actually, the *VFH* unifies two functionalities: moving around obstacles and local navigation. In the concept presented here, these two functionalities are split into two behaviors: the reactive behavior *AO* and the tactical behavior *Look for Corners* which performs a geometrical analysis of the scene.

#### 4.4.4. RB: Look for gaps

The behavior *Look for Gaps* improves the ability of the robot to drive through narrow gaps without slowing down too much. Additionally, various situations are avoided in which the robot could get stuck when the robot enters or leaves the gap in a direction which is more orthogonal than parallel to the gap's middle line.

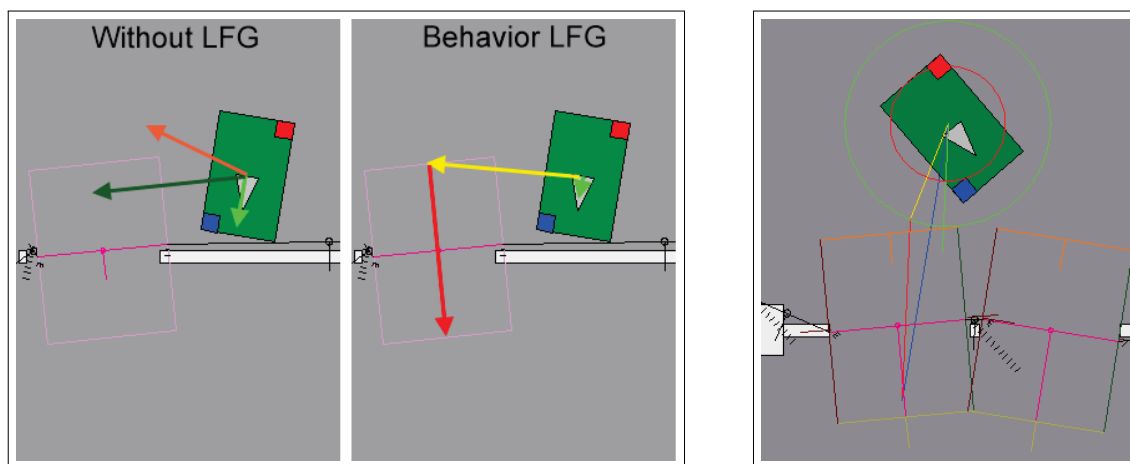


Fig. 4.26.: Sketch and screenshot from MCAGUI illustrating the concepts of the behavior *Look for Gaps*. The gap's area off attention is marked with the pink box, the gap itself with the pink line crossing the gap. The green arrow points towards the target. Left: Without LFG the *AO* behaviors move the robot sideways along the wall. Middle/right: The behavior LFG first generates a vector towards the baseline of the *AoAt* (yellow) and after reaching this location a vector to the center of the baseline on the other side of the gap (red). The target attracting behavior is inhibited simultaneously.

This behavior is activated when gaps are found in the local area and contributes a attracting vector once the robot comes close to a selected gap's defined *area of attraction* (*AoAt*). LFG provides a velocity set-point vector as output and inhibits the target attracting behavior. First, the behavior module generates an

output vector, which points to the center of the baseline of the area of attention, making sure the robot is located sufficiently far away from the corners of the gap to pass safely. This can force the robot to drive backwards a little, for example if the robot is located directly at a wall besides a door. After coming close to the center of the AoAt's baseline the module generates a new output vector which points towards the center of the AoAt's baseline on the other side of the gap and sets the robots orientation accordingly. This way the robot is forced to drive straight through the gap until it has passed the gap completely (see Figure 4.26).

**Quality of Gaps:** A small area in direction of the robot's (intended) movement is searched for gaps e.g. for free room that is framed by objects' corners (acquired from the *object database* of the *LWM*). The width of the free area, e.g. the gap ( $w_G$ ), is compared with the minimum ( $w_{min}$ ) and maximum ( $w_{max}$ ) width a gap must fulfill. Additionally the gap's quality ( $Q_G$ ) is influenced by the angular difference ( $\theta()$ ) of the gap's center ( $\vec{C}_G$ ) compared to the relevant direction ( $\vec{P}_T$  – most probably the direction towards the goal) of the robot, up to a maximum angular displacement. The gap with the best quality is chosen to contribute an attracting vector.

$$Q_G = \frac{\max(w_G - w_{min}; 0)}{w_G} \cdot \frac{\max(\theta_{max} - |\theta(\vec{C}_G, \vec{P}_T)|; 0)}{\theta_{max}} \quad \text{if } w_G < w_{max}, 0 \text{ otherwise} \quad (4.19)$$

**Attracting vector:** The attracting vector ( $\vec{u}_G$ ) points towards the middle line of the gap – first to the starting point ( $\hat{M}_S$ ) then to the end ( $\hat{M}_E$ ) – and has a length of 1 and is then weighted and adapted to the parameters  $f_p$  (a proportional factor),  $m$  (motivation) and  $i$  (inhibition). The better the quality of the gap, the stronger the urge exerted by the behavior to divert from the current course and drive through the gap. In the second version of the behavior it will be added a second phase in which the attracting vector becomes stronger when the gap becomes more narrow.

$$\vec{u}_G = \hat{M}_S \text{ or } E \cdot \frac{\max(w_G - w_{min}; 0)}{w_G} \cdot f \cdot m \cdot (1 - i) \quad (4.20)$$

$$a = \|\vec{u}_G\| \quad (4.21)$$

$$r = \|\vec{Q}_G\| \quad (4.22)$$

**Activity and rating:** The activity ( $a$ ) of the behavior module is the normalized length of the attracting vector. The rating ( $r$ ) is proportional to quality of the gap.

#### 4.4.5. RB: Follow Wall

This behavior is an auxiliary behavior and is not motivated in the usual case but can be motivated for example if requested by the plan which is executed. It provides an additional velocity set-point vector that is useful in corridors. For example a two directional traffic can be implemented this way. The vector points towards a line which is in parallel to a long obstacle e.g. a wall.

#### 4.4.6. RB: Break Tie

This behavior is activated if the ratings of the *Avoid Obstacle* have a high value and the activities a small value for some time or the resulting vector is very small - thus, the robot is stuck. *Break Tie* generates a velocity vector that points along the obstacle to get the robot moving again, strongly amplifying small

preferences. Since using the second version of the *Avoid Obstacle* behaviors the many tests performed have shown that this behavior is not necessary any more.

#### 4.4.7. RB: Orientation of the robot

There are a number of behaviors which control the orientation of the robot. Obviously, these are only applicable when the robot utilizes a holonomic or at least omni-directional drive system or when the robot finally reached the designated parking position. As the behavior modules are straight-forward (besides the orientation part of *Look for Gaps*), they will be only summarized. They all provide a velocity set-point vector in which only the third part ( $\Theta$ ) is used. These vectors are finally merged using a maximum fusion behavior module – thus selecting one – as it does not make sense to sum-up orientations.

##### Orientate along Corridor

<b>Situation:</b>	Can be activated for example by the plan which is executed, usually not motivated.
<b>Data used:</b>	Occupancy map
<b>Objective:</b>	This behavior tries to detect the direction of a corridor based on the occupancy maps and orientates the robot in parallel with the corridor.
<b>Output:</b>	Velocity vector $(-, -, \Theta)$

##### Orientate in Robot's Driving Direction

<b>Situation:</b>	Active when operating in the <i>Autonomous Mode</i> or <i>Guiding Mode</i>
<b>Input:</b>	Current Velocity $(x, y, -)$
<b>Objective:</b>	This behavior orientates the robot with the front in its movement direction.
<b>Output:</b>	Velocity vector $(-, -, \Theta)$

##### Orientate to User

<b>Situation:</b>	Active when operating in the <i>Following Mode</i>
<b>Input:</b>	Position of the user
<b>Objective:</b>	This behavior orientates the robot with the <i>force sensitive handle</i> and the touch screen towards the user.
<b>Output:</b>	Velocity vector $(-, -, \Theta)$

##### Orientate according to special orders

<b>Situation:</b>	Motivated on demand
<b>Input:</b>	Fixed orientation
<b>Objective:</b>	This behavior orientates the robot according to a given orientation. This is for example used at parking positions or the “Turn Around to Support Loading” <i>Force Commands</i> as well as while operating in the <i>Manual Steering Mode</i>
<b>Output:</b>	Velocity vector $(-, -, \Theta)$

#### 4.4.8. RB: Tasks-oriented input behaviors

This group of behavior modules acts as the local source of the control data flow. The individual modules each provide a velocity set-point vector. One of these vectors is selected by a maximum fusion behaviors depending on the current *mode of operation* and task (see Fig. 4.27).

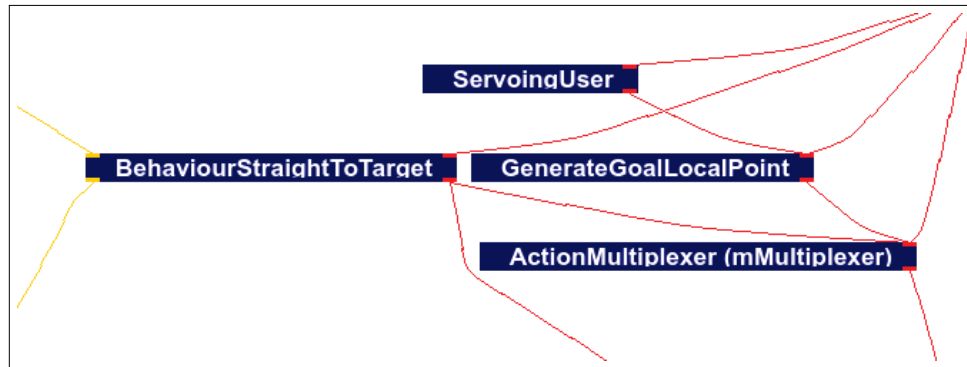


Fig. 4.27.: MCA implementation of the task-oriented input behaviors of the *reactive layer*. The module for manual control is outside this MCA-group and fed into the fusion behavior (the “ActionMultiplexer”) via the “Controller Input” of the group.

### RB/TB: Goal attracting Straight to Target

This behavior module transforms the output of the *tactical layer* which works based on coordinates into velocity set-point vectors as they are used by the *reactive layer*. The output vector point directly towards the goal’s location and usually has a length of 1. When reaching a defined distance to the goal, for example 1m, the length of the vector is reduced proportionally to the remaining distance and a timer is set to a defined value. The vector is reduced to zero when either the robot has reached the goal coordinates (with a small safety margin of few centimeters) or when the timer equals zero. This way the robot always comes to a stop, even when sensor or self-localization fuzziness prevents the robot from hitting the spot exactly – or from knowing that the spot was hit exactly. This behavior is used during the *Autonomous Mode* or *Guiding Mode*.

In the *Guiding Mode* this behavior interacts with “RB: Behaviors for adapting the velocity to the task and the user” (see Section 4.4.2) to provide the guiding functionality.

### RB: Manual Control

While operating in the *Manual Steering Mode* this behavior module transforms input provided directly by the user or an operator into a velocity set-point vector. The data source might be a joystick, teleoperation, or the *force sensitive handle* (see Chapter B.6 “Force sensitive handlebar”). The transformation is straightforward and provides a 3D vector with a length between 0 and 1. This behavior plays a major part in the *control sharing* as described in Chapter 6.6 “Sharing and trading of control”.

### RB: Servoing

This behavior module lets the robot imitate the movements of a moving object as closely as possible. Again, the data sources vary depending on the target object as was described in *RB: Follow Moving Object*. But this time the object’s motion is taken out of the *local world model* instead of the position. The output of this module is again a velocity set-point vector for the robot.

This behavior plays a major part in the *control sharing* as described in Chapter 6.6. It provides the main part of the *Servoing Mode* and can for example be used succeeding the *Guiding Mode* making sure that the robot does not impair the user after guiding him to a designated target.

#### 4.5. The tactical behaviors

This section elaborates on the *tactical behaviors* which make up the *Behavior Network* of the *tactical layer* (see Fig. 4.28), therefore the upper – or more deliberative – part of the *local navigation*. In general, these behaviors accept task or a goal location from “above”, perform a geometrical *scene analysis*, and generate corresponding sub-goals as output. They will be indicated as being “*tactical behaviors*” by including “TB:” in the sections’ titles. The individual behaviors will again be discussed in an bottom-up order - see Fig. 4.29 for the MCA implementation. Coming along with the descriptions are several supplementing sketches and video snapshot from an onboard camera, showing the robot’s perspective on the scene (courtesy of [Ö10]).

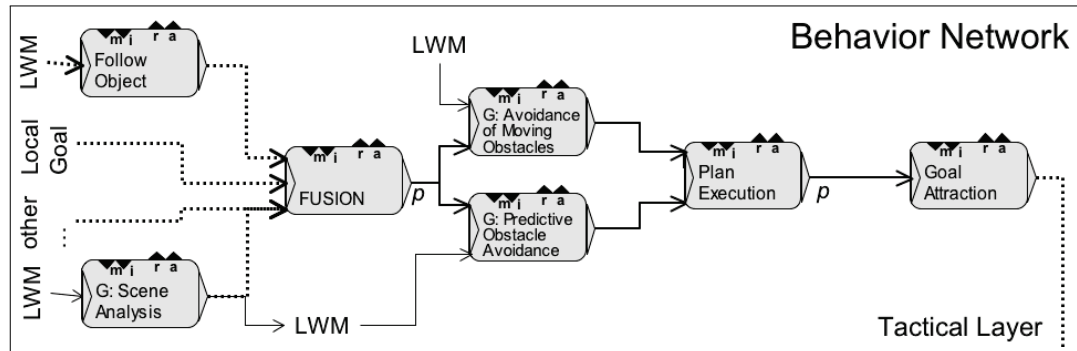


Fig. 4.28.: Abstract view on the *Behavior Network* showing the various behavior groups (indicated by “G:”) of the *tactical layer*. The layer consists of four major steps: input, fusion of inputs, processing, and output. The goal coordinates are the common language among the modules.

Due to the fact that the goal points received from the *global navigation (strategic layer)* are model-based or abstracted from the real environment, the *local navigation* has to adapt these goal points to the actual geometry of the scene. This is a major task of the *scene analysis* which is the topmost group of the *local navigation*. The *tactical behaviors* dedicated to human robot interaction will be described in the corresponding chapter: Chapter 6.5 “Tactical behaviors for adapting to and communicating with the user”.

##### 4.5.1. TB: Look for Corners – geometry-based obstacle avoidance and local navigation

<b>Situation:</b>	The next goal point is not reachable by the robot in a direct path
<b>Input:</b>	Goal point
<b>Data used:</b>	Robot position, robot shape, obstacles ( <i>object database</i> )
<b>Objective:</b>	Generation of sub-goal points that lead the robot around obstacles and enable the robot to avoid U-shaped obstacles / dead-ends.
<b>Output:</b>	First sub-goal point that leads the robot around the obstacle, trigger for speech output if the robot cannot find a path

The behavior *Look for Corners (LFC)* is maybe the most important and surely the most frequently used behavior among the *tactical behaviors*. Compared to the reactive *Avoid Obstacle* behaviors which are limited to their *area of attention*, LFC uses the complete area of the (local) occupancy map and is therefore less prone to getting stuck or to suffer from other problems of local navigation methods. Additionally, it ensures an efficient local path by analyzing the size and positions of obstacles as early as possible. LFC is always used as long a target location is given by higher behaviors or the *global navigation* – therefore the robot is not steered directly for example by the *force sensitive handle*.



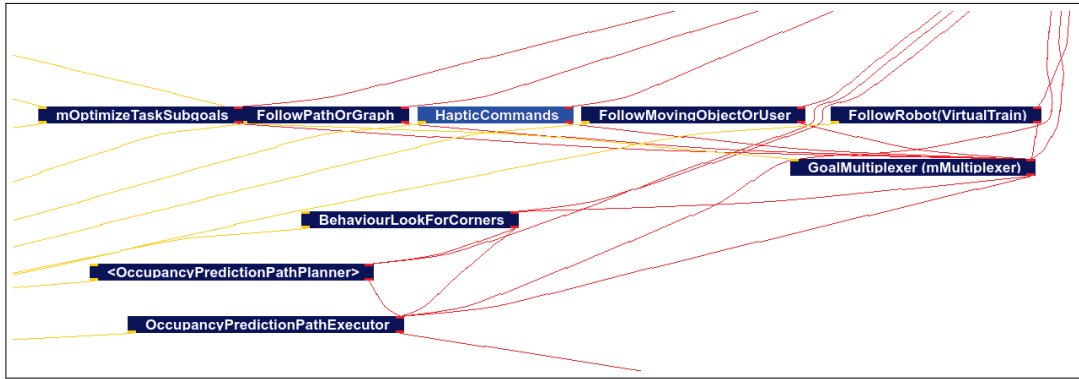


Fig. 4.29.: MCA implementation of the tool chain of the *tactical* layer. The input from several task-oriented behaviors is fused (“OptimizeTaskSubgoals” collects the outcome of the *scene analysis*) and then processed by the geometry-based obstacle avoidance and avoidance of moving objects (see Chap. 5).

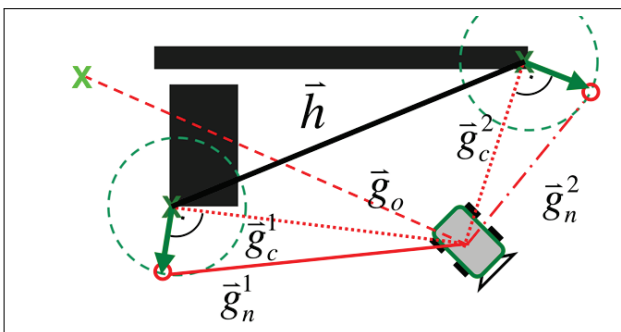


Fig. 4.30.: Geometric construction of the vectors ( $\vec{g}_n$ ) towards the sub-goal candidates (red circles), based on the vectors towards the obstacle’s corners ( $\vec{g}_c$ ). They are placed in a certain distance to the corners of the obstacles equal to the robot’s size (plus a safety margin). The vector  $\vec{g}_o$  points towards the original goal (green X).

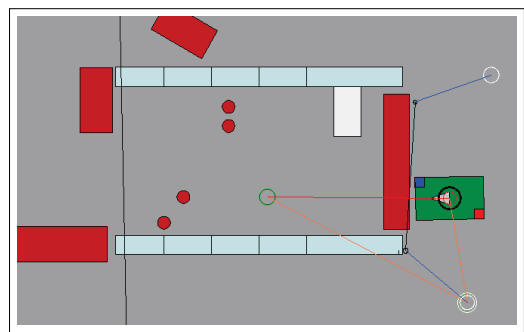


Fig. 4.31.: Sub-goals shown in the MCAGUI: The path to the goal (red line) is blocked by an obstacle (red box). Two candidates are generated at the visible endpoints of the obstacle (white circles). The (currently) best one is chosen as new sub-goal (green circle around white circle).

While driving towards a goal, or when ordered to do so, the location of the goal is compared to the start and end points of the local obstacles from the *object database*, in which all visible obstacles are stored together with a set of characteristic information (see the dedicated paragraph at the end of the description of this behavior). If there is no free direct path (see  $\vec{g}_o$  in Fig. 4.30) of the needed width from the robot to the goal point, all obstacles which block this path are analyzed. For all blocking obstacles first-iteration sub-goals are generated: a vector ( $\vec{g}_c^i$ ) is calculated from the robot towards the obstacle’s end points. Then a second vector orthogonal to the first one is calculated with a length equal to the width of the robot (plus a 20% safety margin). These give the first-iteration goal points. the paths to these are again checked for blocking obstacles (robot to sub-goal and sub-goal to original goal). Blocked sub-goal points are marked as illegal and new sub-goals are generated. This procedure is continued in a recursive manner. In the next step all non-illegal candidates are tested if their “own” obstacle blocks them. In this case they are moved along the circle defining their distance to the obstacle’s end point until they are no longer blocked. Finally the new candidates are ordered by quality (see paragraph below). The best point is chosen as new sub-goal. This way the robot is driving on a local visibility graph. Fig. 4.31 and 4.33 show this concept in an MCAGUI screenshot and in a video snapshot from the robot’s point of view.

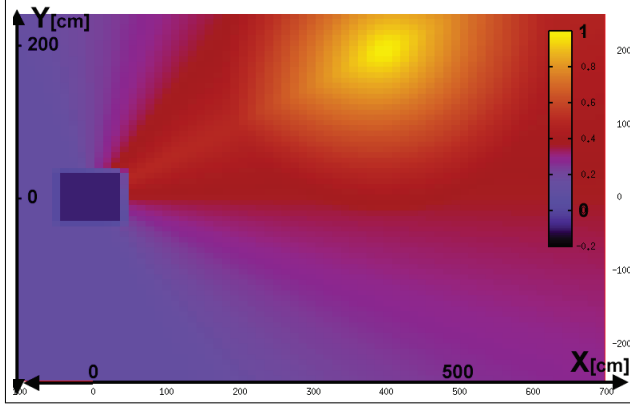


Fig. 4.32.: Plot of the X-Y-plane illustrating the dependence of the candidates' quality regarding their individual position and the position of the target. The robot is illustrated by the dark blue rectangle, the target at the yellow spot with the highest quality. The warmer the color, the higher the quality of a sub-goal candidate located at this position.

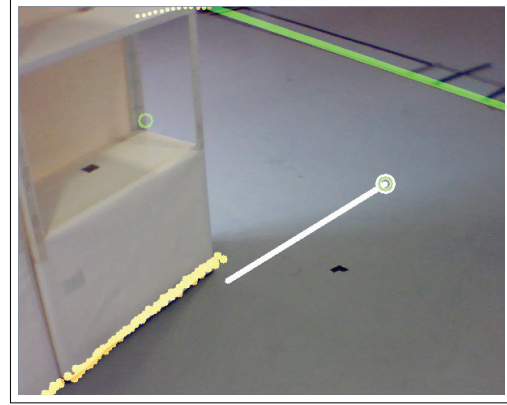


Fig. 4.33.: Video snapshot from the robot's perspective showing the concept of LFC: The goal is the green circle on the left hand side behind the shelf, the robot is diverted to the white circle on the right hand side first.

**Calculation of the candidates' quality:** The candidates for the new sub-goal are ordered by their quality ( $Q_C$ ) which depends on the corners' distance ( $d_{T,C}$ ) and direction ( $\Theta_{T,C}$ ) compared to the target (Fig. 4.32 shows a plot) as well as the width of the free room around it ( $w_C$ ), taking into account a minimum width ( $w_{min}$ ) the robot needs because of its physical dimension. Additional constant parameters are used to set a minimum quality ( $q_{min}$ ) and to limit the quality resulting from a large free space around the corner ( $w_{max}$ ). The best corner is chosen as new sub-target and temporarily replaces the original target.

$$Q_C = \max^2(q_{min}; (1 - \frac{\Theta_{T,C}}{2\pi})) \cdot \max(q_{min}; (1 - \frac{\min(d_{T,C}; d_T)}{d_T})) \cdot \max(\min(w_C; w_{max}) - w_{min}; 0) \quad (4.23)$$

The robot sticks to a chosen goal point until it was reached or until a new sub-goal point is of better quality for one second to prevent oscillations between sub-goal points.

Fig. 4.34 shows that this way the robot is able to move around obstacles independent of shape or size including dead-ends as long as the robot is able to perceive them as an continuous obstacle.

A special situation occurs if there is only one obstacle in the database with the distance between start and endpoint smaller then the robot's width or if no legal candidate is left in the end. Here the geometry-based obstacle handling is considered as failed and the initial goal point is handed down to the *reactive behaviors*. The robot will try to drive towards the original goal, while uttering a speech output to inform the user and other bystanders, hoping that people will free a path for the robot. Additionally, a change of the current perspective might reveal a passage which was not visible before.

#### Output data

$\vec{u}$ : Position ( $X, Y$ ) of the best candidate

$a$ : Quality  $Q_C$  of the best candidate

$r$ : Average quality of the candidates

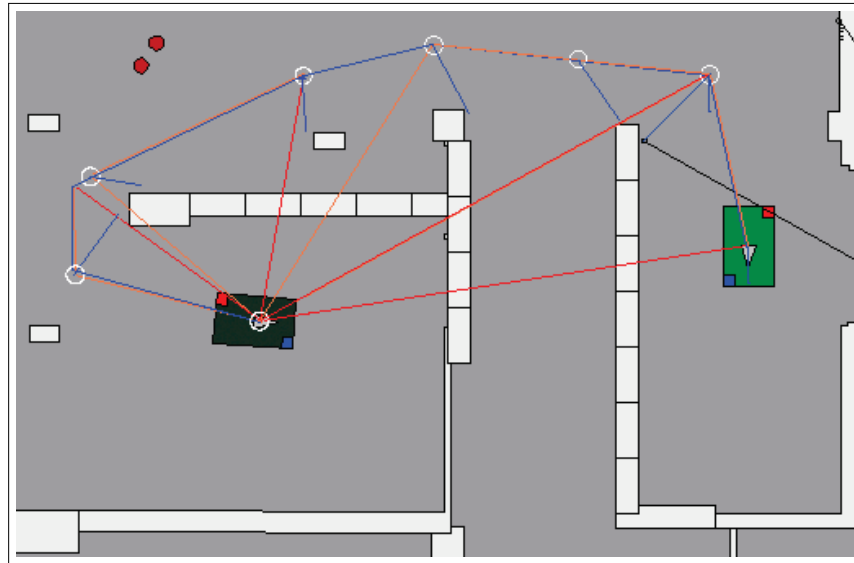


Fig. 4.34.: Sketch illustrating the individual sub-goal points generated on the path leading from the start (green robot shape) to the target point (grey robot shape). It shows how the robot is able to reach a target location even in complex situations (e.g. starting inside a dead end and passing a misleading corridor) only relying on sensor data (therefore without map knowledge).

#### 4.5.2. TB: Follow Moving Object

This behavior module is the first one out of two to follow a moving object – this could be the robot’s user, some other human, or another robot. Accordingly, the input data, namely the position of the object, is provided by several different sources and taken from the *local world model*:

- The *user tracking with onboard sensors* (see Appendix C.2 “Detecting and tracking the user using onboard sensors” and [63], [59])
- The user or person tracking of the *intelligent environment* (see Appendix C.3 “The *intelligent environment*”)
- The shared *local world model* of other robots (see Chapter 7 “Multi-Robot Coordination”)

The behavior operates during the *Following Mode* and simply drives towards the target and stops in a defined distance. For this purpose a velocity vector is generated as output which points directly towards the goal’s location and usually has a length of 1. When reaching a defined distance to the goal, for example  $1m$ , the length of the vector is reduced proportionally to the remaining distance. The vector is reduced to zero when either the robot has reached the goal coordinates (with a small safety margin of few centimeters) or when the timer equals zero. In the *Following Mode* this behavior interacts with “RB: Behaviors for adapting the velocity to the task and the user” (see Section 4.4.2) to provide the following functionality.

This behavior plays a major part in the *control sharing* as described in Chapter 6.6.

#### 4.5.3. TB: Virtual Train

This behavior is the second behavior for following a moving object and obviously operates during the *Following Mode*. In contrast to the first one which just follows the target object, this module tries to generate

a “virtual train”. Thus, this module lets the robot follow the exact path of the target object – as long as the *Avoid Obstacle* behaviors do not intervene. Additionally, some intelligence is implemented for letting the “train” drive backwards, for narrow spaces, and to use shortcuts. Again, the output is a velocity set-point vector.

This behavior is usually used to attach a second (or more) robot to a given one and thus described in detail in the chapter focusing on the multi-robot behaviors: Chapter 7.4 “TB: Virtual train”.

#### 4.5.4. TB: Force Commands

This behavior executes the *Force Commands* given by the user via the *force sensitive handle* such as “Park on the side” or “Turn around to support loading”. These will be presented in the chapter focusing on HRI: Chapter 6.4.1 “Interaction based on force input by the user”.

#### 4.5.5. TBG: Geometrical Scene Analysis – *tactical behaviors* focusing on adapting to the dynamic environment

This group of *tactical behaviors* provides the source of the control data flow of the *tactical layer* for plan-based tasks. The behaviors adapt the goals received from the *strategic layer* to the actual environment – once the needed data is available, e.g. the goal comes within sensor range.

The goal locations received from the *global navigation* are based on models (e.g. product database, *topologic-metrical map*) and apt to be inconsistent with the actual situation of the dynamic environment. Examples of sources for such inconsistencies in the supermarket scenario are:

- Dynamic objects (e.g. a palette or a parked shopping cart) occupy the target
- Dynamic objects (e.g. a palette or a parked shopping cart) block the way to the target
- A self-localization inaccuracy “virtually” moves the model-based target into an obstacle
- Moving objects or people temporarily occupy or block a target
- Dynamically generated target points are illegal (e.g. pointing gestures, inaccuracy in the user position estimation in combination with a “come here” command)
- Errors in the application specific database (e.g. illegal product locations)
- The topological plan is not executable because topological links or areas are blocked by dynamic objects or by re-arrangements of the shop’s structures (which have not been updated in the topological model)

Four major classes of situations of interest can be identified:

- Target points are occupied or are not reachable
- The user has the get access to the target location, not the robot
- The robot encounters obstacles when having to travel in a neighboring *topological area*
- The topological plan is not executable

For these situations of interest several behavior modules were developed which check for the critical situations using the *object database* (see also [Rit10a]). They either modify the goal location before handing it down to the remaining behaviors or trigger feedback for the user, communicating the problem when it cannot be solved in the *tactical layer*. The objective is to rely as less as possible on model data but on the real environment instead.

### TB: Goal Point Adaptation

This behavior module becomes active when the goal location comes within the robot's sensor range. The module checks if there is a sufficient amount free space around the goal location for the robot to park. Otherwise the goal is altered according to the obstacles.

<b>Situation:</b>	The area surrounding a goal point is occupied by an obstacle or the goal is too close by the obstacle
<b>Input:</b>	Goal point
<b>Data used:</b>	Occupancy map, and <i>object database</i> , and shape of robot
<b>Objective:</b>	Displacing the goal point so that it can be reached by the robot
<b>Output:</b>	New goal point

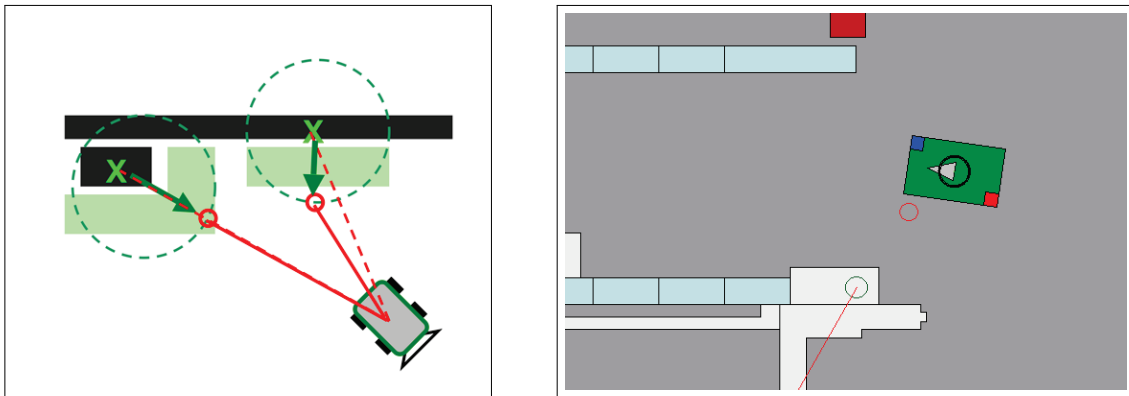


Fig. 4.35.: Goal Point Displacement: In a dynamic environment goal points (green x or green circle) can be blocked. If the space surrounding the goal location is occupied by an obstacle or that close by an obstacle that the robot cannot reach the goal, the goal point is displaced.

Sketch: The free room (light green area) in the close vicinity of the original goal is analyzed and a new goal (red circle) is generated in a suitable distance to the obstacles. In cases of wall-shaped obstacles the goal is moved orthogonally to the obstacle, in all other cases towards the robot.

Screenshot: MCAGUI screenshot showing an exemplary scene with the original goal (green circle) and the new goal (red circle).

If it was identified that the goal point lies within an obstacle but closely to the surface, or the goal point lies that close to an obstacle that the robot cannot reach the goal, the local occupancy grid map is analyzed to find a new suitable position for the goal point. In the general case the goal point is moved away from the obstacle towards the robot to a point where the smallest distance to the closest occupied grid cell is larger than half the robot's width (with a 20 % safety margin) (see Fig. 4.35 the left hand side sketch and the MCAGUI screenshot). The right hand sketch of Figure 4.35 shows a special case: keeping in mind that the control system was designed to operate a robot in a supermarket, most of the goal points will be product positions at shelves. Therefore, if the goal point is located within (or too close by) an wall-shaped obstacle

(long straight line in the occupancy grid) the goal point is not moved towards the robot but away from the obstacle orthogonally.

### TB: Identifying Parking Positions

This behavior module was designed for reaching a target together with a user. If a user is involved, usually not the robot shall reach the target, but the user. In the supermarket scenario the target might be the spot in front of the shelf containing a desired product. To not impair the user, the robot shall drive past the goal a little bit so that the user can easily reach it. This could be handled in the plan, but then the exact direction of approaching the goal would have to be included in the plan as well, relying even more on model data. Here the opposite strategy is taken: model data is replaced by real world data when approaching the goal.

<b>Situation:</b>	The robot has to reach a product in <i>Guiding Mode</i>
<b>Input:</b>	Target position (maybe already be moved by other behaviors)
<b>Data used:</b>	Occupancy grid, <i>object database</i> , and shape of robot
<b>Objective:</b>	Finding a suitable parking position which enables easy access to the goal for the user
<b>Output:</b>	New goal point

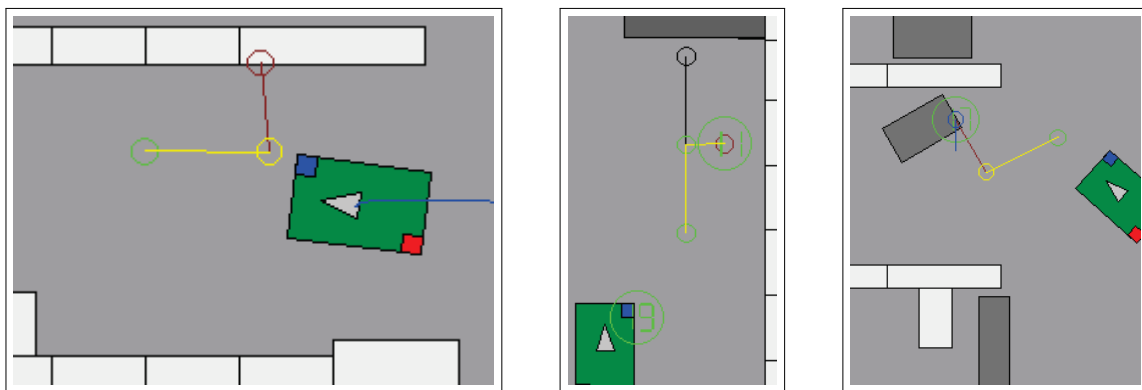


Fig. 4.36.: Left: Standard situation – the robot chooses a parking position a little more than half a robot length behind the goal to enable easy access to the product located in the shelf. The brown line shows the target re-direction by the *Goal Point Adaptation* described in the preceding section (because the target lies within the shelf). The yellow line with the green circle indicates the re-direction to ease shopping for the user.

Middle: The position behind the product which the robot should have taken is blocked. So the target location is redirected half a robot length in front of the product. The user has to move around the robot but still has free access to the product.

Right: The target is occupied by an object. The target is moved to the free room and a parking position is found.

If the target position is a location to reach while guiding a user (e.g. a product in the supermarket) the precise target location is adapted (Fig. 4.36): basically the robot will move to a target position so that the user can easily reach the goal while not unnecessarily blocking the corridor. Therefore, the robot moves past the goal instead of stopping in front of it. Special situations occur when the goal is at the end of an alley: this is either a dead end and the robot cannot move past the product, or it is an intersection and the robot would block the crossing alley. In these cases the robot stops before the product – the user has to go around the robot but can still reach the goal easily. This behavior is usually combined with the *Goal Point Adaptation* from the preceding section which makes sure that the target can actually be reached by the robot.

### TB: Generation of Topological Navigation Points

This behavior module enables the robot to pass into a neighboring *topological area*. The plan from the *strategic layer* provides only the center point of the link between two *topological areas*. Where exactly to pass this link is identified by this module, taking into account the robot's position, the succeeding goal, and local obstacles.

<b>Situation:</b>	The robot shall move into an neighboring <i>topological area</i>
<b>Input:</b>	Topological link to move past, succeeding goal (target location or next link)
<b>Data used:</b>	Robot position, robot shape, obstacles ( <i>object database</i> ), free space (occupancy grid)
<b>Objective:</b>	Generation of sub-goals which lead the robot towards and past the topological link. These sub-goals shall take the robot's position and the succeeding goal into account to generate an efficient path. Of course, these points have to have enough distance to obstacles so that the robot is able to actually reach them.
<b>Output:</b>	Two sub-goal points, one on each side of the link

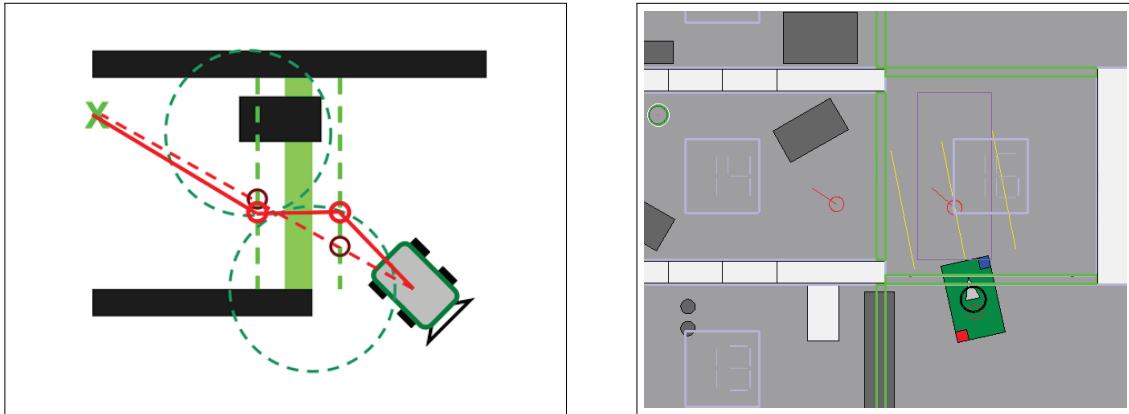


Fig. 4.37.: Topological navigation points: Goal points (red circles) leading the robot into the next *topological area*, past the topological link (green rectangles), are generated to enable the robot to execute the plan of the *topological navigation*. In a dynamic environment the links between the *topological areas* can be (partially) blocked so the *topological navigation* points have to be adapted to the free room in the close vicinity of the link.

Left: Sketch illustrating the geometrical construction. The line from robot to the goal is intersected with lines in parallel with the link. The intersections give the first iteration goals – the represent the shortest path. Afterwards circles with the radius equal to the robot's width are drawn around the obstacles' corners. The first iteration points are moved along the lines until the are not located inside the circles.

Right: Screenshot from the MCAGUI showing the points leading the robot from area No. 16 to No. 14. The navigation points are adapted to the succeeding goal (green circle at the left hand frame) and the obstacle behind the link.

Geometrical methods are used to lead the robot past the link, to avoid obstacles close by the link and to generate an efficient path regarding the robot's position and the succeeding goal. The behavior *Look for Corners* would be able to move the robot around the obstacles – but it would not enforce that the robot passes the topological link. In contrast, the behavior presented here enforces the plan of the *topological navigation*. For this purpose, two auxiliary goal points are generated, as depicted in Figure 4.37: one in front of and one behind the link.

Each, before and after the link two parallel straight lines are generated in a defined distance in parallel to the link. The intersections of these two lines with the line from the robot to the succeeding goal are

calculated. These are the first iteration sub-goal points. They meet the requirement of the efficient path. Then for each obstacle a circle is calculated around the closest corner where the radius is half of the robot's width (plus 20% safety margin). The first iteration points are moved along the lines so that they are not situated inside such a circle. If this is not possible, the median of the intersecting area between the circles is taken. This gives the final (second iteration) sub-goal points.

If the link should not be passable at all this is handled by another behavior: *Detection of Blocked Topological Links and Areas*. Figure 4.41 from the next section shows a video snapshot showing the robot's view: the light green circle on the left is the first iteration goal, the darker circle is the second iteration goal point.

### TB: Detection of Blocked Topological Links and Areas

This behavior module analyzes if a topological link which is to be passed is actually passable, or not. The result is given back upwards to the *strategic layer* which will update the *topological map* and start re-planning, if necessary. Additionally, a corresponding speech output to the user is triggered. This behavior cooperates with the behavior *Generation of Topological Navigation Points* as it is not able to move the robot itself. The behavior for generating the navigation points will move the robot towards not visible areas of the link.

<b>Situation:</b>	A topological link has to be passed
<b>Input:</b>	Topological link (ID, position, width)
<b>Data used:</b>	Shape of the robot, <i>object database</i> , occupancy map
<b>Objective:</b>	Identify if the link is passable or blocked
<b>Output:</b>	State of the link, trigger for speech output

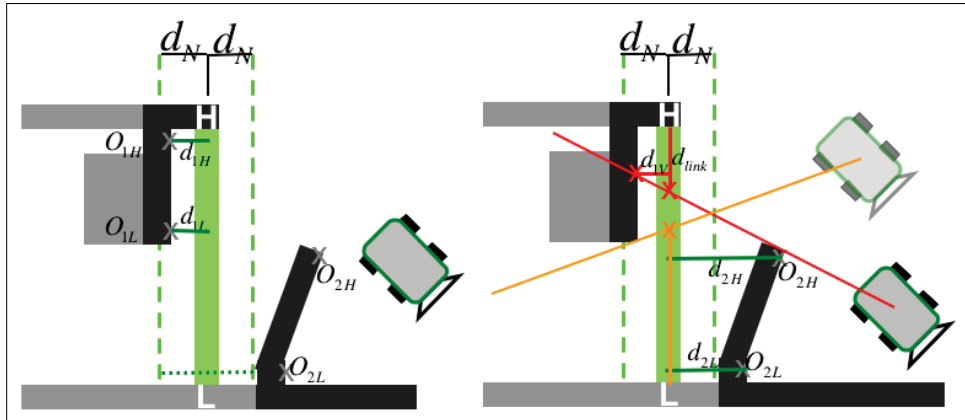


Fig. 4.38.: Detection of blocked topological links: The obstacle on the (H)igh side of the link (green rectangle) blocks the link, the obstacle on the (L)ow side does not block the link.

Left: The robot cannot detect a passage from its current perspective. Therefore the link is marked as partially blocked on the high side and as potentially blocked from a low side perspective.

Right: After moving forward the robot detects that there is a passage. If  $d_{2H}$  or the distance between  $O_{1L}$  and  $O_{2H}$  would have been smaller than  $d_N$  the link would have been marked as blocked on the high and low side therefore the *topological map* would have been updated resulting in a re-planning.

For the *topological navigation* it is necessary to be informed about impassable topological links to be able to update the *topological map* and to re-plan the current task. A geometrical analysis of the *object database* is performed (see Fig. 4.38). It has to be taken into account that the local situation can differ based on the perspective of the robot: While the link can seem to be blocked from a perspective from the (L)ow



side of the corridor it can be obviously passable from a (H)igh side perspective. A video snapshot showing the robot's perspective can be seen in Figure 4.41 and some screenshots of the MCAGUI show results in Figures 4.39 and 4.40.

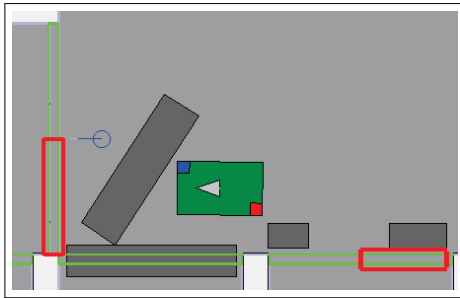


Fig. 4.39.: Two links are marked as partially blocked (red box on the link (green boxes)), the left one is additionally marked as MBL (which is not illustrated because the visualization uses the red box for both flags).

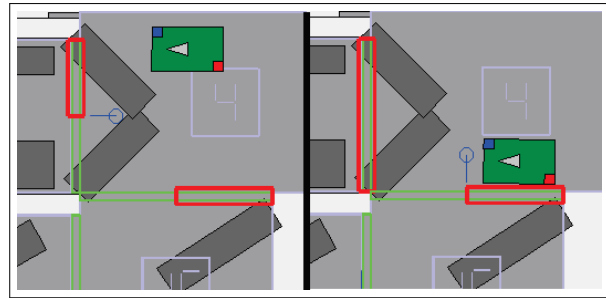


Fig. 4.40.: The robot starts on the upper side of the left hand side link and marks it with MBH. After traveling along the link the MBL flag is set as well, completely blocking the link.

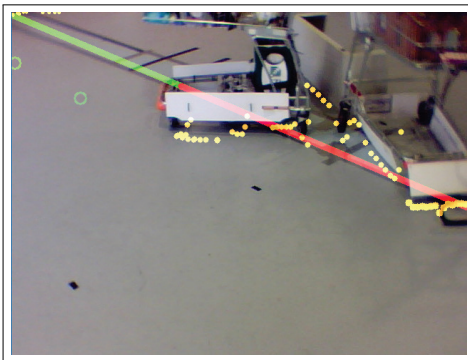


Fig. 4.41.: Video snapshot from the robot's perspective showing a partially blocked barrier as indicated by the red line. The green line overlaying the RFID barrier indicates the free part of the link. The green circles indicate the *topological navigation* points described in the previous section.

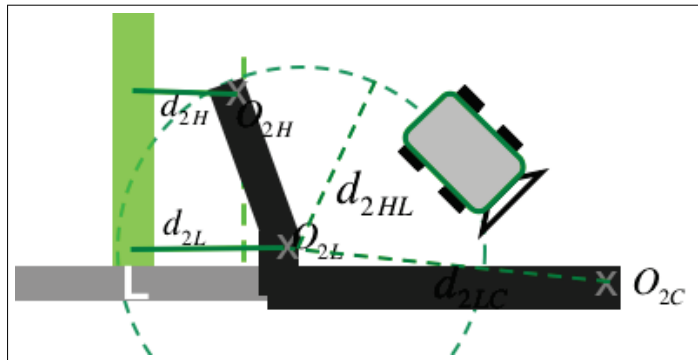


Fig. 4.42.: Special Situation: the gap at the lower end of the link is wide enough for the robot. An estimation is necessary if the robot can move around the obstacle on the lower side. This is done based on the position of the obstacles end point ( $O_{2C}$ ).

To be able to take care of links being partially blocked or seem to be blocked from a certain perspective four flags are introduced, corresponding to the orientation of the link (which depends on the numbers of the neighboring areas, but has no further effect here) :

- PBH: Link is partially blocked on the high side
- PBL: Link is partially blocked on low side
- MBH: Link maybe blocked seen form a high side perspective
- MBL: Link maybe blocked seen from a low side perspective

The link is marked as blocked if (see Fig. 4.38):

1. Both PBH and PBL are set and the distance between  $O_{1L}$  and  $O_{2H}$  is smaller than the distance needed by the robot to pass ( $d_N$ )

#### 4. BBC: Navigation, Obstacle Avoidance and Safety

2. PBH is set and the distance from  $O_{1L}$  to the low side of the link is smaller than  $d_N$
3. PBL is set and the distance from  $O_{2H}$  to the high side of the link is smaller than  $d_N$
4. Both MBH and MBL are set.

The PBx flag is set when the link is obviously blocked on the corresponding side. If an obstacle is detected on the high side, two points ( $O_{1H}$ ,  $O_{1L}$ ) are identified: the ones closest to the high and low side of the link which are in front of or behind the link (not besides the link). The distances between these and the link are measured: if  $d_{1H}$  and  $d_{1L}$  are smaller than the needed space of the robot ( $d_N$ ) the PBH flag is set (the same for the low side accordingly – both  $d_{2H}$  and  $d_{2L}$  are  $> d_N$  so the PBL flag is not set).

The MBx flag is set when the robot cannot see a passage past the link from the corresponding perspective but it is not sure that the link is blocked: If an obstacle is detected on the low side, two additional points ( $O_{2H}$ ,  $O_{2L}$ ) are identified: the ones closest to the high and low side of the link which are in front of or behind the link (not besides the link). Both  $d_{2H}$  and  $d_{2L}$  are  $> d_N$  so the link is not partially blocked on the low side. To determine if the robot can find a passage the field of view (FOV) is analyzed (red line): if the distance between the intersection between the FOV-line and the link's end ( $d_{link}$ ) is  $> d_N$  and the distance between the first obstacle on this line and the link ( $d_{1V}$ ) is  $> d_N$  there is a passage. If one is smaller no passage is found (as in the sketch). But as  $d_{2H} > d_N$  there might be a gap which the robot cannot see. Therefore, the MBL flag is set. (The same is done for the high perspective accordingly –  $d_{2H}$ ,  $d_{link}$  and  $d_{1V} > d_N$  from this perspective (orange lines) so the MBH flag is not set – the link remains passable).

A special situation is sketched in Fig. 4.42:  $d_{2L} > d_N$  so the robot would be able to pass the barrier at the low side. But can the robot actually reach this gap? Now we additionally take the end of the obstacle ( $O_{2C}$ ) into account. If  $d_{2HL} > d_{2LC}$  we estimate that the robot cannot pass around the obstacle and therefore the link is marked with the PBL flag.

#### TB: Generation of Virtual Topological Areas

This behavior module analyzes if a *topological area* is completely blocked. If this is the case the module provides output for the *strategic layer* resulting in splitting the *topological area* into two virtual areas and the re-planning of the current task. Additionally the user is informed.

<b>Situation:</b>	The robot shall move to any kind of goal point in the current topological area.
<b>Input:</b>	Goal point
<b>Data used:</b>	Robot position, robot shape, obstacles ( <i>object database</i> )
<b>Objective:</b>	When the robot shall move to any kind of goal point in the current topological area and the path is blocked the behavior searches for gaps in the local environment. If no gap of sufficient size is detectable the area is marked as blocked and split along the obstacles into two topological areas.
<b>Output:</b>	Position and size of cut between areas, signal for topological re-planning, trigger for speech output to user

As this behavior does not include building a global map (which would be against the general concept), it obviously operates robustly only if the areas are not too large: the two sides bordering the *topological area* have to be inside the local occupancy map. The aim is to detect blocked corridors, not to detect blocked wide spaces or halls. Basically, this module works in the same way like *Detection of Blocked Topological Links and Areas* mentioned before, just taking lines across the area instead of topological links. Figure 4.43 shows an example in an MCAGUI screenshot.

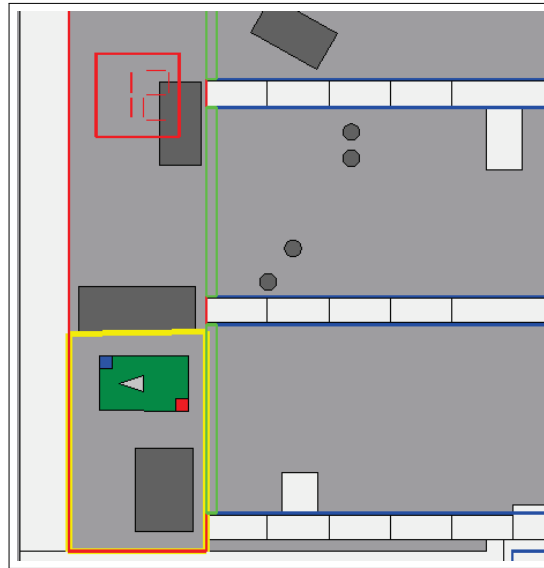


Fig. 4.43.: Virtual areas: The robot is stuck in area No. 12 (red box) while trying to traverse it from the bottom part to the upper part. The reason is an obstacle that appeared there (dark grey box) which blocks the area. As result, the area is split up into the initial area 12 and a virtual new area (yellow box). To reach the upper half of area No. 12, the robot will have to re-plan and leave the virtual area using the right hand side link. The other areas are marked with blue boxes, the topological links in green.

#### 4.6. The *strategic layer* – global navigation and planning

After describing the *reactive layer* and the *tactical layer* making up the *Behavior Network*, this section will describe the planning and the *topological navigation*. Commands are received through the *command interface* (see Chapter 3.3.3 “The *command interface*”), processed, and handed down to the *tactical layer* as goal location. There the majority of the control system’s capabilities is provided by the *Behavior Network*. In parallel, events are collected and processed. Some events result in speech output to the user such as when a target is reached or when re-planning is needed. As the focus of this thesis lies on the *Behavior Network* and this layer applies standard methods only and keeping in mind that three more behavior-focused chapters are following, the description is kept brief. Extensive information can be found in the mid-study thesis of F. Stehle [Ste09a].

As motivated in the introduction of this work by human behavior, the planning and global navigation should be very coarse and use highly abstract data only – the *topologic-metrical map*. The *topologic-metrical map* is actually a *topological map* whose areas (or nodes) are enriched with metrical information of the relative positions of the links to neighboring areas. Using flexible task trees to divide and refine tasks (in the style of *flexible programs* by S. Knoop [95]), a given task is split into sub-tasks. For each sub-task, a plan is generated based on the *topological map*. Finally metrical goals are generated which represent sub-targets such as links between topological areas. These targets are handed down to the *local navigation* described in the two preceding sections.

The main idea here is to abstract the task and the corresponding model-based plan from the environment to be independent from ordinary-scale changes in the dynamic environment. The downward interface consists of model-based goal points or topological navigation points according to the topological links between areas. The obvious risk here is that abstracted goal points can be unreachable for the robot in a dynamic

environment. The goals are not meant to be reached exactly but are meant to be a fuzzy target. The *local navigation* therefore treats the goal points as suggestions only and utilizes several behaviors to adapt them to the actual environment, which is the responsibility of the geometrical *scene analysis*.

In addition to planning and *topological navigation*, this layer manages the *modes of operation*. Mode changes are requested by the *communication layer* through the *command interface* or result from events from the behaviors such as reaching a goal. The *strategic layer* stores the current *mode of operation*, performs the mode transitions and orchestrates the behaviors accordingly, in particular by (de-) activating the ones for task- and user-adaptation.

#### 4.6.1. Topologic-metric map

The environment – in the chosen lead scenario a shop (Fig. 4.44) – is split up into several areas. Figure 4.45 shows an examples for such a partitioning. The areas and their connections are mapped to a *topological map* without local metrical maps (Fig.4.46). The only metrical information needed for a certain area are the relative positions of the entry- and exit-points which link the individual areas to each other. These metrical coordinates are defined relative to the coordinate systems of the individual areas and afterwards stored as metrical annotation with the areas – giving a *topologic-metric map* (see Chapter 2.3.2 “Maps and mapping” for an distinction).

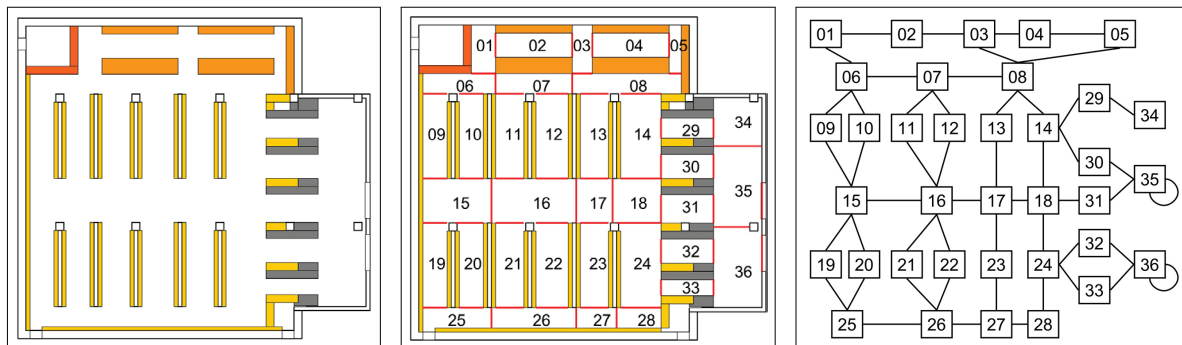


Fig. 4.44.: Geometric map of a shop.

Fig. 4.45.: Map divided into areas

Fig. 4.46.: Resulting *topological map*

In addition to the metrical annotations, special information can be stored for each node, including semantic annotations (Fig.4.47). The semantic information can be used as an indication how an area should be traversed (i.e. crossing a room or following the wall of a corridor). Additionally, these information can tell whether a node should be preferred or avoided in a global plan, can include directions how to reach the next node from the current entrance of the actual node, restrictions to velocity or acceleration, or certain dangers which wait in this area like a grate in the floor which would render an optical motion sensor useless or a mirror that could trick a camera system. Furthermore, temporal information like a blockage of a corridor or the prohibition to enter a room or other warnings might be of interest. The information can be updated online by the robot to inform other robots about knowledge it has acquired.

#### 4.6.2. Global planning

Based on the *topologic-metric map* with semantic annotations, the global route planning is performed. Given a new task by the application specific component, the new task is inserted into the task tree on the first

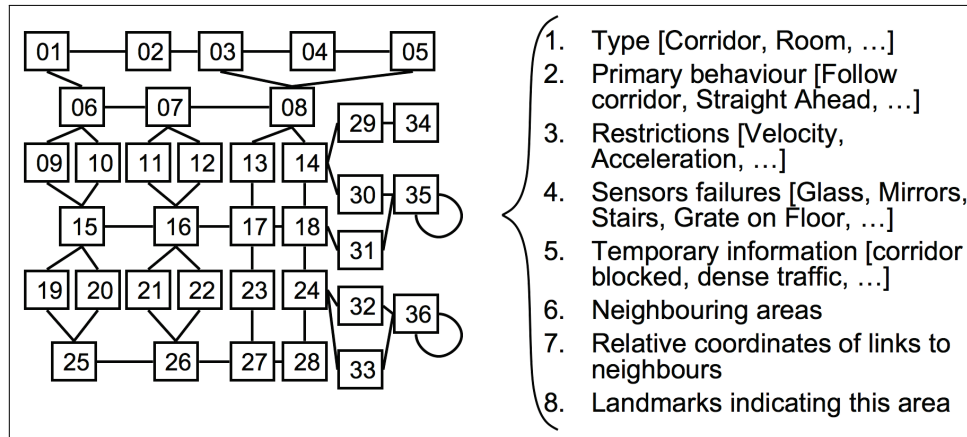


Fig. 4.47.: Topologic Metric Map: A *topological map* with metric and semantic annotations

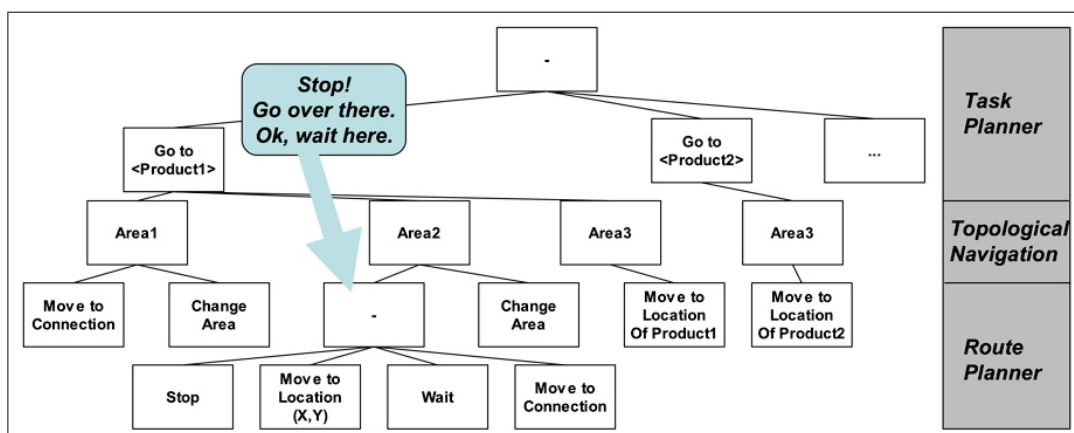


Fig. 4.48.: A plan is generated in three levels: task, topological, and route. The leaves of the tree-like structure are atomic commands which can be executed by the lower layers. The flexibility of the concept allows interruptions of the plan and even postponing whole sub-trees. This is indicated by the blue balloon representing sudden commands by the user.

level. For each navigation task the robot plans a route on the *topological map* from its current topological node to the node that contains the target. An  $A^*$  algorithm is used for this planning step. The planning algorithm is enhanced in a way that it is able to utilize the semantic information provided, for example to generate constrains or weights out of them to influence the plan. Because the *topological map* is a very high level of abstraction, the navigation takes place on a small state space. Therefore the processing costs are very low and re-planning is fast and can be done often, if necessary. The chosen nodes make up the second layer of nodes in the task tree, as illustrated in Figure 4.48. Based the topological nodes, a list is generated that contains the relative coordinates of the links needed to leave or enter the individual nodes as well as other auxiliary tasks such as “wait”. In the end the relative coordinates of the target in the last area are added to the list. This list gives the bottom layer of the task tree – the leaves. The leaves of the task tree, containing the metrical navigation sub-tasks, are then handed down to the *tactical layer* one by one. As shown in the figure, the task tree can be altered at run time: new nodes can be inserted, obsolete ones can be deleted, and nodes can even be expanded into new sub-trees. If for example the user gives the “stop” command, which is received and handed down by the *communication layer*, this command is inserted into the there at

the current point of execution, postponing the current action. A complete sequence of tasks can be inserted this way – in the illustrated example the user interrupts the robot and gives the command to move to a local position. Afterwards, the robot will continue to travel towards the link to the next topological area.

#### 4.6.3. Evaluation of global planning

The *global navigation* was indirectly tested alongside many other tests and experiments and proven successful in hundreds of times. Additionally, a stress test with the real computation hardware of *InBOT* was performed (see Fig. 4.49). The *topological maps* have been automatically generated because they did not exist in the real world in the needed size. But as only the topological planning process should be tested, this does not influence the result of the test. Maps with 10, 100, 1000, 4000, and 8000 square-shaped topological areas have been generated with random links between adjacent areas. The planning process was always successful. It took less than 200ms on maps with less than 1000 regions – keeping the intended size of 5m to 10m for an area in mind this would be a square of 150x150m to 300x300m or 25k – 90k square meter. On the map with 8000 regions the planning for a 16km topological path took 7 seconds, which is too long regarding a waiting user. But a map of this size is very unlikely in the considered applications.

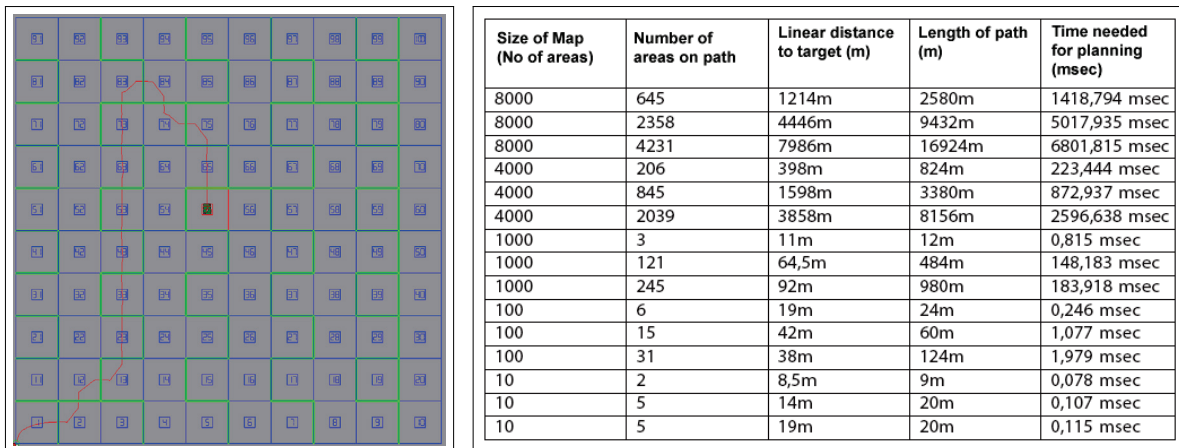


Fig. 4.49.: Stress test of the *topological navigation*: The table shows the maximum, minimum, and average results from computing random routes (topological nodes and corresponding metrical sub-goals) on huge random maps with random links between the areas. The figure on the left shows an exemplary map including the path the (simulated) robot has taken. The computation of the routes has been performed on the real hardware of the robot.

#### 4.7. Application logic and *communication layer*

As introduced in the beginning, this layer collates the application-specific functions of the control architecture, facilitating a strict distinction of application and core of the control system. The *communication layer* houses some kind of *application logic* and translates between the user and the robot's control system. In the supermarket scenario it has four main functionalities focusing on the user interface:

- Receiving commands from the user (speech and/or touch screen) and transforming them into commands to the robot's control system
- Receiving events from the robot's control system and transforming them into output to the user (speech and/or touch screen)

- Housing the product database of the shop including the products' relative position in the *topological area* they are contained in
- Supporting the shopping process by managing shopping lists, recipes, and so forth

More information on these modalities can be found in the corresponding section in the HRI-related chapter (Chap. 6.2). The *communication layer* communicates with the *strategic layer* using the *command interface* as described in Chapter 3.3.3 “The *command interface*”.

As the alternative implementations of the *communication layer* are not part of the core part of the control system, they are introduced in the Appendix. Two implementations have been used with *InBOT*: For this thesis the *InBOT-UI* (see Appendix C.4) was developed, implementing speech output, a bar code scanner, as well as a shopping list and recipe management via touch screen. In addition, a second *communication layer* was integrated, namely the *CR-UI* (see [87] and Appendix C.5), developed at TU Vienna.

## 4.8. Experiments and evaluation

Following the description of the individual components of the navigation system (including the individual components' evaluation), this section presents three tests which have been conducted using the complete system – the last two tests even involving users recruited “from the street”. The focus of this section will be on the test results regarding the navigation system. In the chapter focusing on HRI (Chap. 6) the tests will be picked up again with an human-factors point of view.

### 4.8.1. First system test

One example out of the tests performed with the complete system is presented here, supplementing the results provided in the individual sections dedicated to the individual behaviors or functionalities, respectively. For the sake of reproducibility, this test has been performed in a simulated environment – the real world tests involving users are described in the subsequent sections. The simulation of the robot platform is very detailed and accurate down to each individual scan point of the laser scanners. Additionally, the simulation – for which large pieces could be gratefully taken from previous work – incorporates the (de-) acceleration capabilities, the drive system and the S300 laser scanners including the corresponding precisions and variances. The simulation does not interfere with the remaining control system, as it just substitutes the *Hardware Abstraction Layer*, using exactly the same interface.

Figure 4.50 shows this test: The robot was given the tasks to visit a list of products and was impaired by lots of obstacles. All components described in this chapter have been used, from the *communication layer* down to the safety reflexes. The robot passed dead-ends, narrow passages, and evaded large and tiny obstacles. The bottom part of the figure shows the robot's knowledge of the experiment: the *topological map*, the local sensor readings, and the path the robot has taken.

### 4.8.2. Second system test

This first large-scale test performed with untrained users was a collaborative activity with KTH Stockholm and TU Vienna. It aimed at gathering first impressions on the HRI system in general and the shopping list assistant (commanded by the *CR-UI*) in particular, thus, the setup was reduced to short linear runs and the

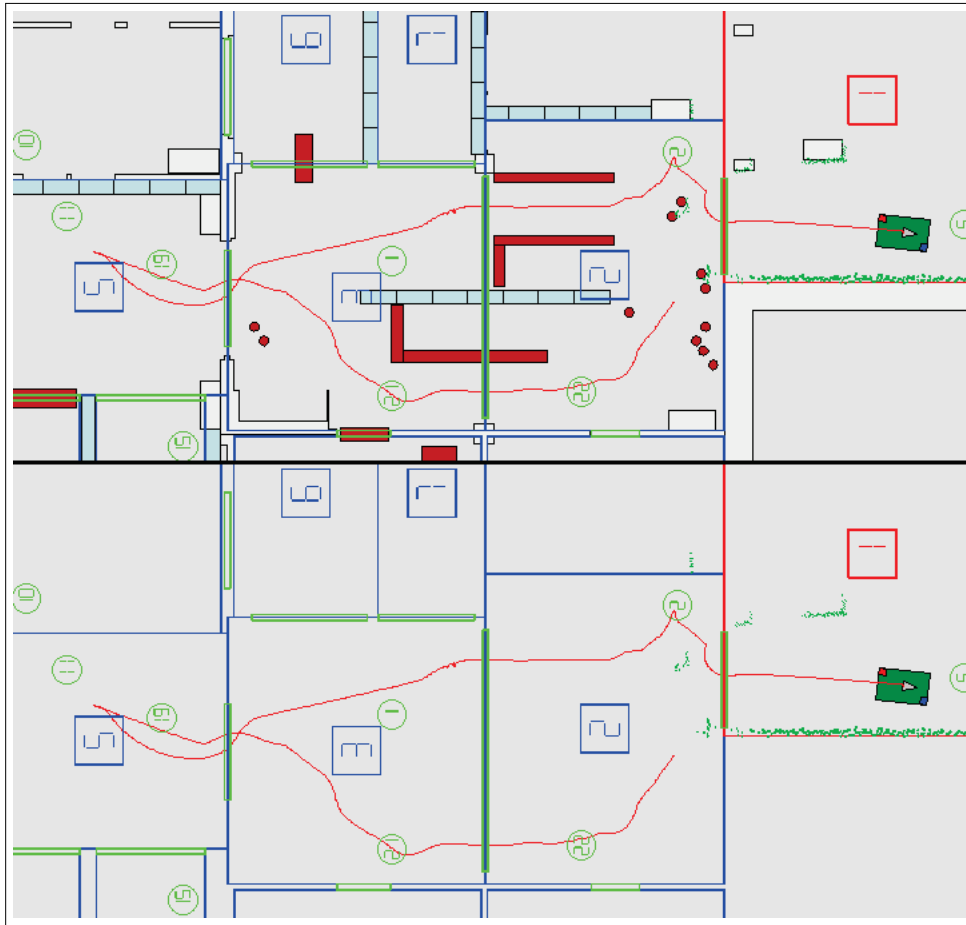


Fig. 4.50.: This figure shows a test run performed with the robot using the MCA simulation environment. The robot was given the task to visit a list of products (located at the places marked with the numbers 21, 19, 11, 2, and 5). The top figure shows the (simulated) real world including lots of obstacles. The bottom figure shows the robot's knowledge. According to the defined requirements, the robot's knowledge is limited to the current sensor readings (green dots) and the *topologic-metrical map* which is indicated by the blue lines (borders of topological areas) and the green rectangles (locations of the RFID barriers). Even though relying only on this few information, the robot successfully reached all products and evaded a large number of obstacles, including a dead-end.

available functionalities were reduced to a basic set. The results are briefly summarized in the Figure 4.51 (top).

#### 4.8.3. Third system test

This third test was the second test with users – again a collaborative action with KTH and TUW. It was performed with full complexity and functionality (omitting moving obstacles). In particular, this time the geometrical *scene analysis* has been used.

The test was performed with 2+10 untrained users. Two pilot runs have been performed a priori to adapt parameters of the navigation system to the environment. Otherwise, the system was identically in all twelve runs. Initially, the users had to enter a scripted shopping list containing 12 products using the touch screen user interface implemented by the *CR-UI*. Then they had to order the robot to guide them to the first nine products. The last three products should be accessed by driving the robot in the *Manual Steering*



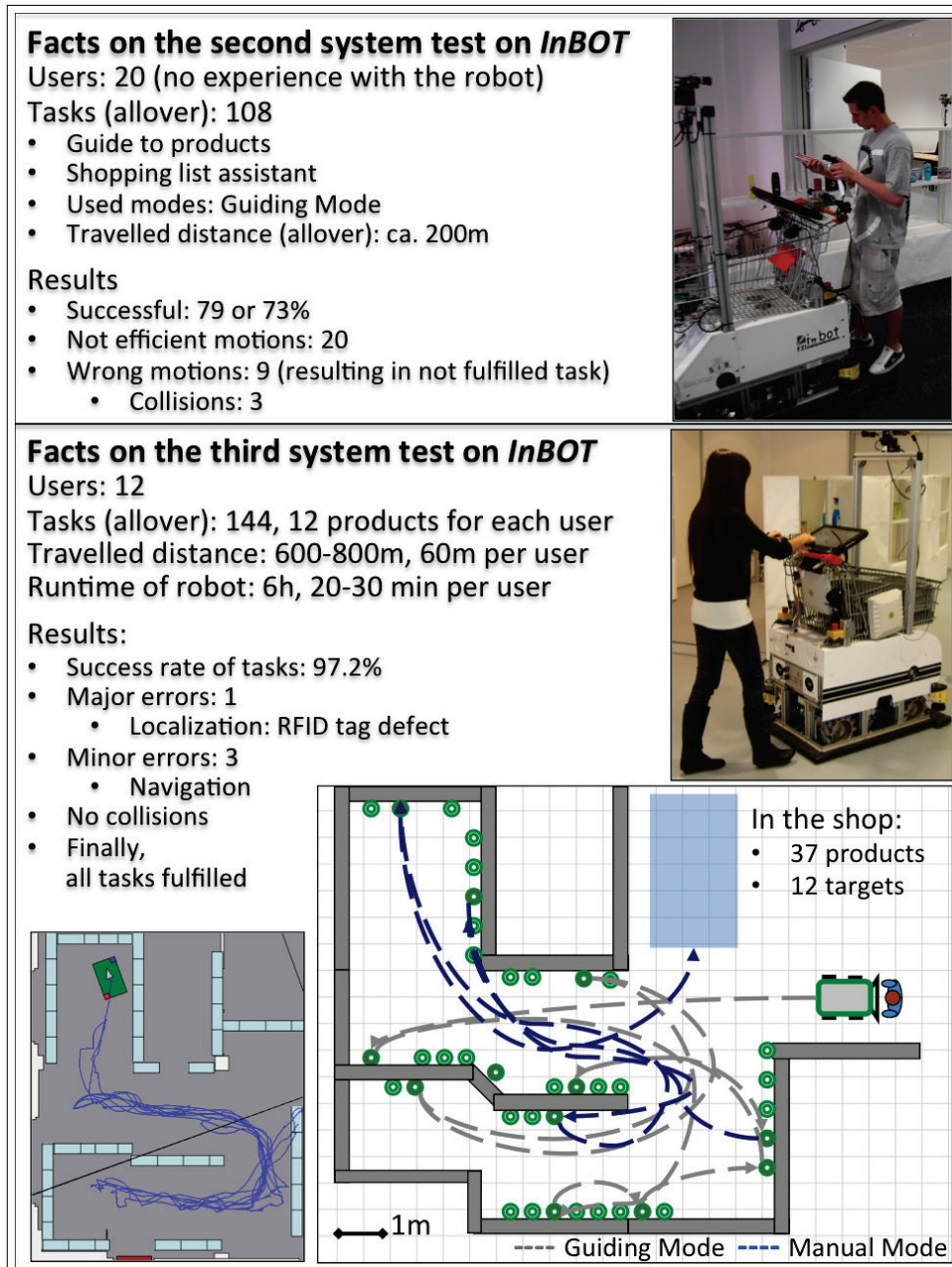


Fig. 4.51.: Facts on the second and third system test on *InBOT* conducted with users.

*Mode.* While maneuvering in a narrow corridor the user had to avoid a box which was dropped in his path in cooperation with the obstacle avoidance assistant. A sketch of this setup is provided together with a summary of the results in Figure 4.51 (bottom).

From a navigation and control system point of view, the test can be considered successful: all users were able to finish all tasks and we observed no collisions at all and hardly any wrong movements. The success rate of the navigation tasks was 97.2% with only 3 errors: one time the robot moved in an incorrect direction. Most probably the corridor *InBOT* should drive into seemed to be blocked (by the user and a person from the experimental staff steering a cart with a camera mounted on top). Hence, the predictive obstacle avoidance tried to find another way. But this could not be verified without stopping the experiment, thus, it counts as

mistake. In the two other cases the robot did not start moving even though the speech output “I will guide you to ...” was uttered. In all three cases the user could solve the problem by the canceling and repeating the command. On the second try the robot executed the task correctly.

#### 4.9. Discussion

When developing the navigation system, the ability focused on was the safe and reliable navigation, thus, the avoidance of all kinds of static obstacles as well as finding efficient paths through cluttered scenes. Inspired by motion behaviors identified when observing humans, a hierarchical approach was chosen. Besides an abstract *topologic-metrical map*, the robot only needs information it is currently able to acquire with its own sensors. Thus, in contrast to methods found on many “guiding robots”, but the graph-based method used by *Robox*, the presented navigation system does not depend on a precise and up-to-date metric global map.

Summarizing, the navigation concept provided in this thesis proposes a three layer approach, inspired by the motion behavior of human beings as described by S. P. Hoogendoorn: the bottom part is a *reactive layer* containing a safety method, restricting the maximum allowed velocity. Also in the *reactive layer* a method based on occupancy grids and virtual force vectors is found. The next part is a geometric scene analysis, where the main component works comparable to local visibility-based methods: The obstacles are scanned for corners and sub-targets are placed at the corners in a save distance. And finally the global part is provided by a *topological navigation*. Figure 4.52 shows that the concept defined by the control architecture has actually been implemented by the navigation system, and Figure 4.53 shows a summary of the MCA implementation of the individual groups and modules.

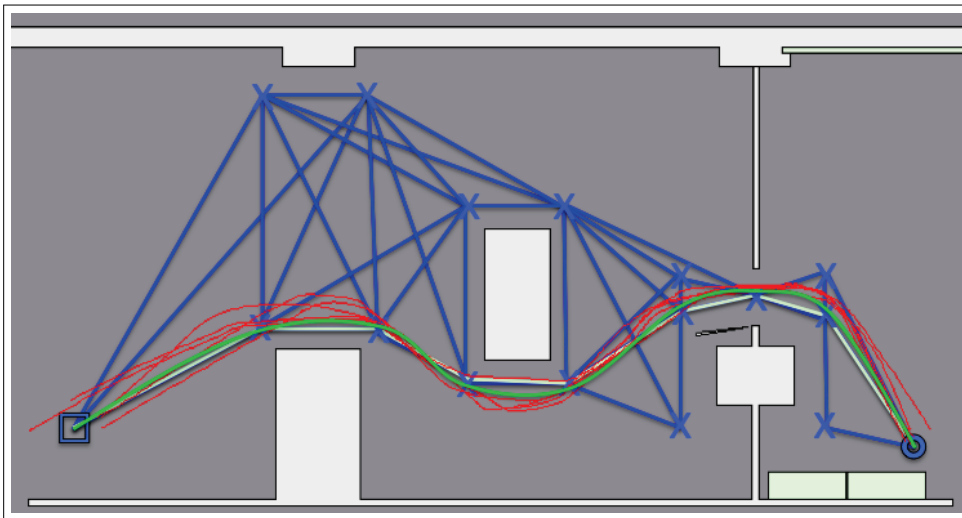


Fig. 4.52.: Comparison of the mesh-based model to the robot’s actual motion: The edge mesh (dark blue) described in the SoA shall model the motion of pedestrians and provide a guideline for the development of the navigation system’s behavior modules. The light blue line shows the selected path between two hot spots according to the model and the green line illustrates the expected robots motion. Finally, the red lines show paths actually taken by the robot *InBOT* when ordered to drive from positions around the blue square to the blue circle.

**Reactive safety:** The reactive safety group splits the environment into several sectors just like done by the *VFH*. In contrast to the *VFH* the polar histogram is not updated using the occupancy grid but based

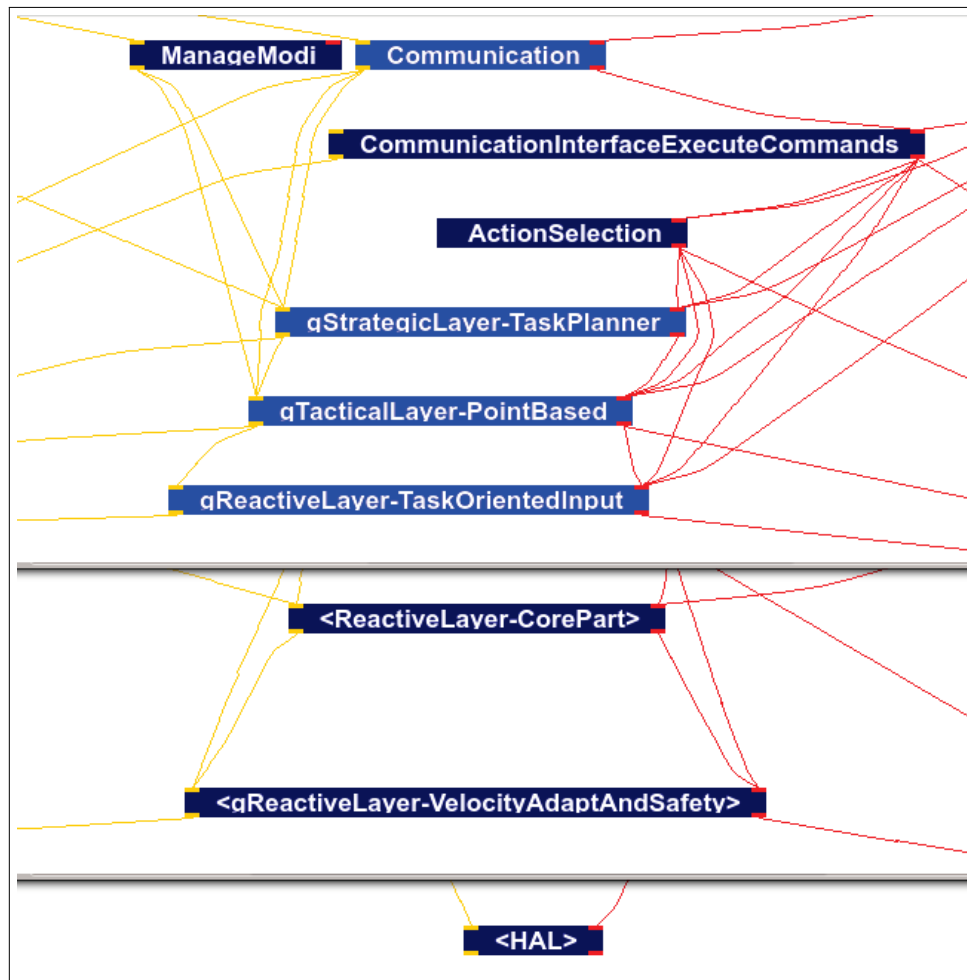


Fig. 4.53.: Summary of the control architecture as it was implemented in the MCA framework. Individual groups of modules were shown in the previous sections and more will be shown in the subsequent chapters. Groups of modules are indicated by “g”, threads encapsulating groups with “<>”. The red lines on the right indicate top-down and the yellow lines on the left bottom-up data flow.

directly on (polar) sensor readings to reduce reaction time. Relevant is the closest sensor reading in each sector, only. Based on the sectors and the robot’s shape (*InBOT* is not circular) a second polar histogram is computed which defines the maximum allowed velocity for each possible direction – down to zero in the worst case.

**Reactive obstacle avoidance:** In contrast to many vector field and potential field methods, the robot’s velocity is taken into account. Furthermore, no “field” is actually computed, but one representative repelling vector only, saving computational power and focusing on the most critical point of the obstacle. Dynamic repulsive vectors are computed and merged with target attracting and other virtual force vectors. A difference to the *VFF* methods is that one representative repelling vector is generated for every object – not for each individual cell. This vector is proportional to the relevance of the complete obstacle, therefore by its most critical cells compared to the robot’s current velocity vector and target direction. The size and density of the obstacle (i.e. the amount of occupied cells) is not relevant. In contrast, when using *VFF* all cells are considered but the cells’ weight is proportional only to the cell’s distance to the robot.

**Predictive obstacle avoidance:** The first two purely reactive methods are supported by a more deliberative method which is a local visibility-based approach. It enables the robot to perform a geometric obstacle analysis prior to the application of the force vectors. Here sub-targets are generated at the obstacles' corners in order to move the robot along an efficient path through the obstacles and to avoid for example moving into dead-ends. As this method does not depend on radial sectors like the *VFH* it enables the robot to use all gaps found between obstacles, independent of their angle. For example it is possible to move into gaps which are radial to the robot. This method could be applied recursively like in *VFH\**, but it is not done here as the behavior module utilizes the complete local occupancy grid and it does not have a global metrical map available.

**Conclusion:** Several specialized behaviors for a hierarchical *Behavior-Based Control* were developed for a robot with an holonomic drive system. Another design feature was the extensibility of the control system. New modules providing new features, like the avoidance of moving obstacles, shall be included easily.

In all tests the reactive components were able to avoid collisions with static obstacles reliably (the handling of moving obstacles will be described in Chapter 5). The predictive obstacle handler generates efficient paths that are comparable to those generated by visibility graph methods, avoiding driving into dead-ends. It should be kept in mind that the robot has no global map knowledge and therefore is only able to plan the path in visibility range of the sensors or within a small local memorized area. The network character of the control system facilitates extending the system with new functionalities. This is done either by straight forward hooking in new behavior modules using the fusion behavior modules or by recombining present functionalities by activating the corresponding behavior modules. This way it is for example possible to use the obstacle avoidance functionality to augment the steering functionality of the *force sensitive handle* so that the intelligent trolley moves around obstacles while it is being steered manually by its user.

## 5. Avoiding Collisions with Moving Objects

After describing the control architecture and the navigation system in the previous chapters, this chapter will introduce a hierarchical approach for avoiding collisions with moving objects. It describes the three behaviors – two reactive and one plan-based one – which are situated in the *reactive layer* as well as in the *tactical layer*.

**Scope of this chapter:** The focus is on the avoidance of accidental collisions with moving obstacles such as customers with shopping carts who are hurrying down a corridor being distracted by the products in the shelves or talking with each other. These behaviors do not aim at avoiding collisions with people who actually try to hit the robot: as the maximum velocity and acceleration of the robot is limited, this would not be possible. Even if the robot's velocity would be sufficiently high, it would not be appropriate in a populated environment having a robot making fast and (for bystanders) unpredictable evasive motions. The *reactive behaviors* could be used this way, as their reaction time is very fast, but on the opinion of the author this would not be appropriate.

In case of conflict between different behaviors, the safety behaviors are of highest priority: on the author's opinion it is more important that the robot does not run into people than that people cannot run into the robot.

**Overview and organization of this chapter:** The challenge of avoiding moving objects – they are not called “dynamic obstacles” here (see Chap. 2.3.3 for an explanation) – is performed in a three-step approach: the lowest behavior is a reflex which moves the robot directly away from moving obstacles, enabling the robot to regain a safety distance. This is of importance when either an object came too close accidentally or suddenly started moving. On top of the first one a second behavior lets the robot free the predicted (local) path of an approaching object, combining an sufficiently intelligent behavior with a suitable reaction time. These two *reactive behaviors* are elaborated on in Section 5.1. To solve complex trapping situations the behavior-based components are topped by a *tactical behavior* (Sec. 5.2) which uses data provided by a local spatio-temporal planner. This planner generates a safe and efficient mid-term path at the cost of less fast reactions. Additionally, the planner can only be applied if the robot is operating in a mode where a target location is given – for example it cannot be applied during *Servoing Mode* and only with limitations in the *Following Mode*. Figure 5.1 illustrates the developed hierarchy and the integration into the remaining control program. The chapter is concluded with a section providing test results (Sec. 5.3) and a discussion of the achievements (Sec. 5.4).

This chapter does not include a description of the data sources for the behavior modules. The necessary information, e.g. the position, velocity, and predicted path of moving objects, will be assumed to be provided by external components. An example can be found in the Annex of this work: the “The *intelligent environment*” (Appendix C.3) developed to support this work for evaluation purposes. It uses cameras and optional laser range finders distributed in the environments to detect and track moving objects. A alternative

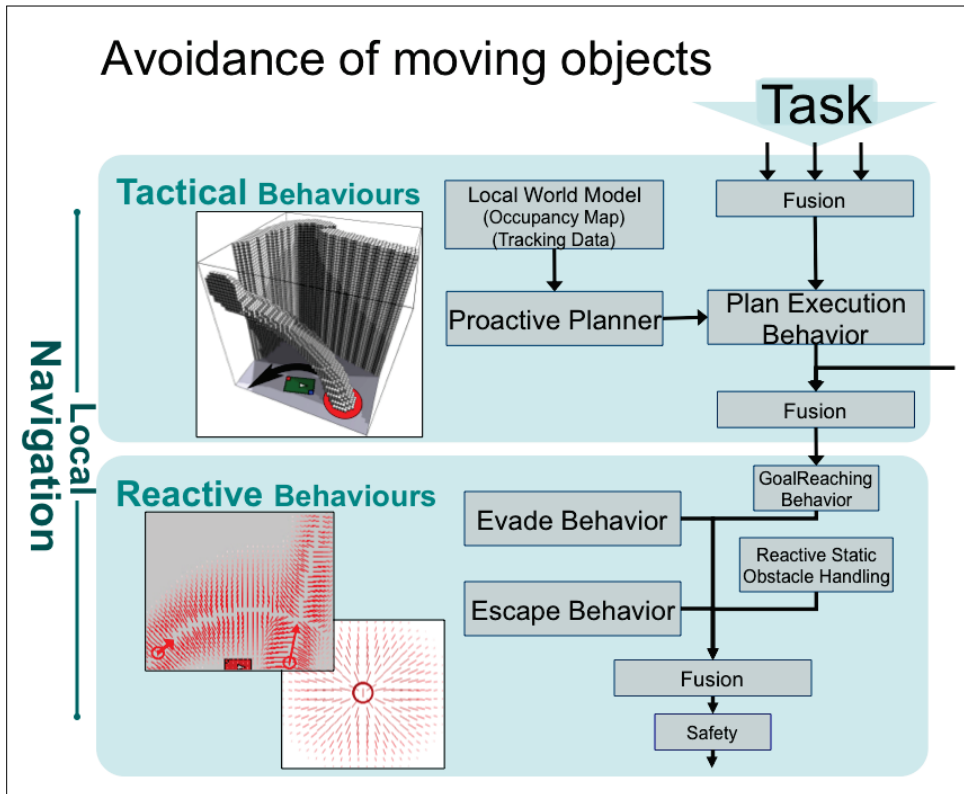


Fig. 5.1.: Hierarchy of the modules responsible for avoiding collisions with moving obstacles: Among the *tactical behaviors* there is a proactive planner which can generate sub goals according to a generated plan. Among the *reactive behaviors* there is one behavior that lets the robot move out of the path of the object and one (emergency) behavior which tries to gain a safety distance by moving straight away from the object. In contrast to the planner, the latter two are fully operable even if no goal point is given but a target direction only, as for example when using manual control, tele-operation, or servoing.

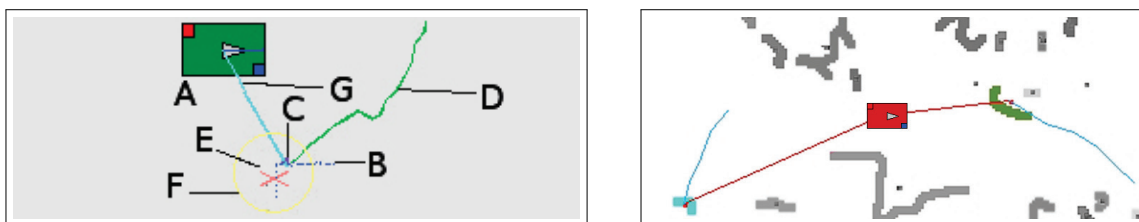


Fig. 5.2.: Exemplary data on a tracked object.

Left: A: the robot, B: the object as seen by the laser range finders, C: the object's center, D: past positions of the object, E: (red X) predicted next position of the object, and F: most probable area of finding the objects center (current time step).

Right: Two tracked objects in the robot's occupancy map captured with the on-board laser range finders.

version using the on-board laser range finders has been used as well (*tracking of moving objects with on-board laser range finders* (see Appendix C.1 “Object tracking based on the occupancy grid and planar laser scanners”). The result is a list of objects which are probably moving, their current position and velocity including the current direction, and their track (past sets of data with corresponding time stamp (see Figure 5.2)). Additionally, a circle segment is matched to the track to determine if the motion is straight or a curve, and in case of a curve to determine the radius. The data is provided via the *local world model*.

## 5.1. Reactive avoidance of moving objects

This behavior group is part of the *reactive behaviors*. It consists of two individual behavior modules that both generate a repelling vector to be merged with the remaining vectors (such as goal attraction or avoidance of static obstacles) which finally add up to the resulting velocity set-point vector (see Fig. 5.3 for the MCA implementation). The first behavior of this group is a safety reflex that generates a vector that points directly away from a nearby moving object to obtain a safety distance. This behavior can be applied even if hardly any information on the object is available. The second behavior needs a movement model from the *local world model* gained by mid-term observations of the object. Based on the movement model a probabilistic temporal repelling vector is calculated – in the following figures a complete field is shown for illustrative purposes, but in the control system actually only the vector relevant for the robot is calculated. Both behavior modules work based on the *object database* containing the detected moving obstacles in robot’s close vicinity along with their characteristics such as position and movement model. The concepts were implemented in F. Steinhardt’s diploma thesis: [Ste09b].

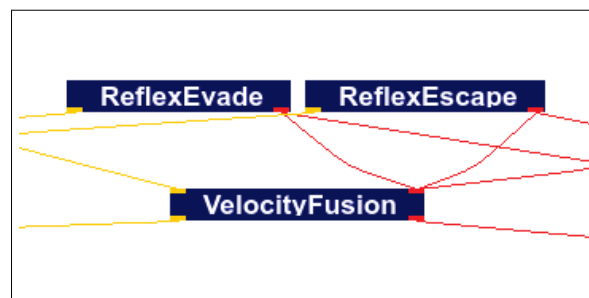


Fig. 5.3.: MCA implementation of the group for avoiding collisions with moving objects. The fusion behavior is the same one which accepts the input from the static collision handling behaviors. Hence, one single output velocity set-point vector is created for all *reactive behaviors*.

### 5.1.1. RB: The Escape Behavior

This behavior module lets the robot retreat directly away when in the the close vicinity of a moving object. To calculate the repelling velocity vector ( $\vec{u}_{Escape}$ ), a repelling vector ( $\vec{R}_O$ ) for each visible moving object ( $O$ ) is calculated. The direction of the repelling vector is from the moving object towards the robot ( $\hat{OR}$ ) and

its length depends on the distance between object and robot ( $|\vec{d}_{OR}|$ ) as well as on the velocity of the object ( $|\vec{v}_O|$ ). The area of influence is restricted by the maximum radius  $r_{max}$ .

$$\vec{u}_{Escape} = \vec{F}_{R_O} \cdot f_p \cdot m \cdot (1 - i) \quad (5.1)$$

$$\vec{F}_{R_O} = \begin{cases} \vec{S}_R & \text{if } |\vec{S}_R| \leq 1 \\ \hat{S}_R & \text{otherwise} \end{cases} \quad (5.2)$$

$$\vec{S}_R = \sum^o (\vec{R}_O) \quad (5.3)$$

$$\vec{R}_O = \hat{O}R \cdot \frac{\max(r_{max} - |\vec{d}_{OR}|; 0)}{r_{max}} \cdot \frac{|\vec{v}_O|}{v_{max}} \quad (5.4)$$

$$a = |\vec{u}_{Escape}| \quad (5.5)$$

$$r = |\vec{F}_{R_O}| \quad (5.6)$$

The moving direction of the object is not relevant. The objective of this behavior module is to keep the robot away from moving obstacles as they tend to be unpredictable. Furthermore, this module is able to operate even when no prediction of the object's motion is available. But obviously the parameter  $r_{max}$  has to be set to a reasonable value to curb the area of attention.  $0.5m$  have been proven to be a suitable value.

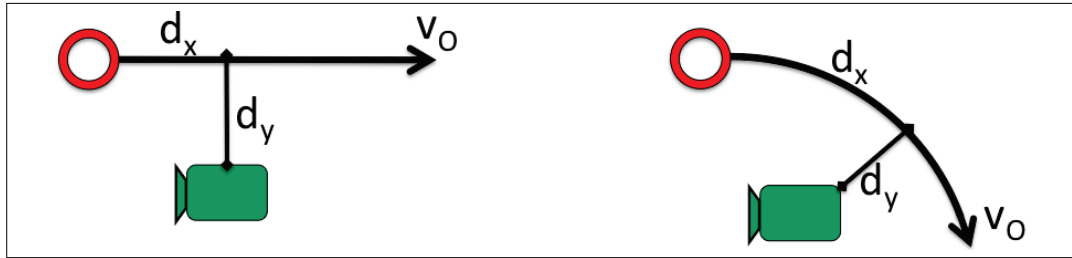


Fig. 5.4.: A moving object (red circle) passes by the robot (green). Two examples for distance vector ( $\vec{d}_O^M = (d_x, d_y)$ ) are illustrated based on the object's estimated motion. Left: the estimated movement is linear. Right: the estimated movement of the object is a curve. In both cases two components are calculated: the first one along the movement direction, the second one orthogonal.

### 5.1.2. RB: The Evade Behavior

This behavior module clears the way of approaching moving objects. A probabilistic estimation of the object's path is calculated based on the estimated movement model of the object. Based on the resulting temporal estimation of the object's location, the repelling velocity set-point vector is calculated, which lets the robot move away from the potentially occupied space and therefore makes a collision improbable.

To calculate the resulting repelling velocity vector ( $\vec{u}_{Evade}$ ) for the behavior module a repelling vector ( $\vec{R}_O$ ) for each visible moving object is calculated. The direction ( $\hat{v}_{O\perp}$ ) of this object-dependent repelling vector is orthogonal to the movement direction of the object ( $\hat{v}_O$ ). The length of the vector depends on the actual velocity of the object ( $|\vec{v}_O|$ ) and a force factor ( $f(\vec{d}_O^M, \vec{V}_O)$ ). The force factor is a function of the distance vector between robot and the model-based movement estimation of the object ( $\vec{d}_O^M$ , see Fig. 5.4) as well as the variance ( $\vec{V}_O$ ) of the movement model. It consists of two components, one based on the distance along the path of the object (the X-component) the other one based on the distance between the robot and the path of the object (the Y-component). Both components are limited by corresponding distance restriction



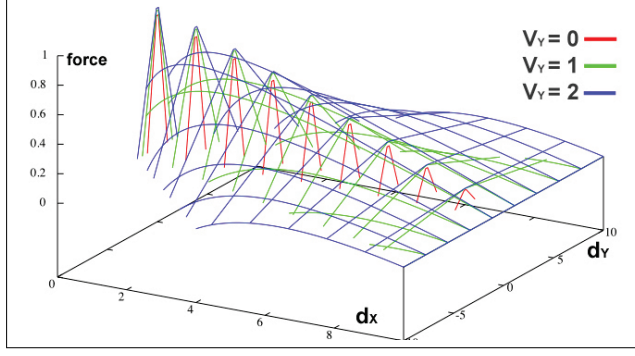


Fig. 5.5.: The complete field of influence of one moving object for three different Y-variances of the movement model. The width of the area of influence in Y-direction depends on the Y-variance ( $V_Y$ ) of the movement model. The length depends on the velocity of the object (the object moves along the X-axis).

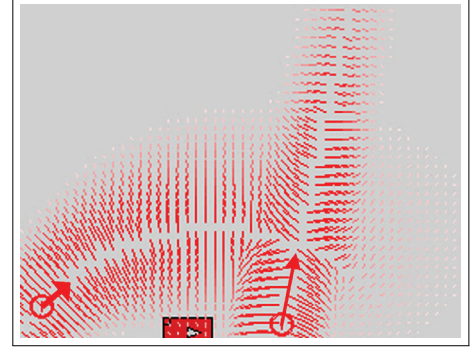


Fig. 5.6.: Two moving objects (circle) approach the robot (box). The various red lines illustrate possible repelling vectors. The object moving in a curve is harder to predict resulting in a raised variance and thus in a larger area of influence.

functions ( $D_X^{max}()$ ,  $D_Y^{max}(V_Y, d_X)$ ). These mainly depend on the parameter for the maximum distance of influence ( $d_X^{max}$ ,  $d_Y^{max}$ ).  $D_Y^{max}(V_Y, d_X)$  also takes the variance of the estimation and the distance along the path ( $d_X$ ) into account. Therefore, the area of effect enlarges with the distance to the object and with the variance (see the plot in Fig.5.5).

$$\vec{u}_{Evade} = \vec{F}_{Ro} \cdot f_p \cdot m \cdot (1 - i) \quad (5.7)$$

$$\vec{F}_{Ro} = \begin{cases} \vec{S}_R & \text{if } |\vec{S}_R| \leq 1 \\ \hat{S}_R & \text{otherwise} \end{cases} \quad (5.8)$$

$$\vec{S}_R = \sum_o (\vec{R}_O) \quad (5.9)$$

$$\vec{R}_O = \hat{v}_{O\perp} \cdot f(d_O^M, \vec{V}_O) \cdot \frac{|\vec{v}_O|}{v_{max}} \quad (5.10)$$

$$f(d_O^M, \vec{V}_O) = \left(1 - \frac{\min(d_X; D_X^{max}())}{D_X^{max}()}\right) \cdot \left(1 - \frac{\min(d_Y; D_Y^{max}(V_Y, d_X))}{D_Y^{max}(V_Y, d_X)}\right) \quad (5.11)$$

$$a = |\vec{u}_{Evade}| \quad (5.12)$$

$$r = |\vec{F}_{Ro}| \quad (5.13)$$

$$d_O^M = \begin{pmatrix} d_X \\ d_Y \end{pmatrix}, \vec{V}_O = \begin{pmatrix} V_X \\ V_Y \end{pmatrix}, D_X^{max}() = d_X^{max} \quad (5.14)$$

$$D_Y^{max}(V_Y, d_X) = d_Y^{max} \cdot (V_Y + 1) \cdot ((V_Y \cdot d_X) + 1) \quad (5.15)$$

### 5.1.3. Experimental results

This section presents some experimental results, showing the functions and the limitations of the *reactive behaviors*. More extensive results on will be given at the end of this chapter when the complete system is evaluated. Fig. 5.6 shows the area of influence – thus the amount of possible repelling vectors – generated by the two behavior modules *Escape* (circular part) and *Evade* (oval part). Here one out of the two objects

## 5. Avoiding Collisions with Moving Objects

does not move straight ahead but in a curve. Additionally, the prediction of one object is bad, so the variance of the movement model rises. Figure 5.7 shows a scene with a shopping cart approaching from the front while Figure 5.8 presents the resulting evasive motion of the robot. The robot is repelled from the predicted path of the objects and moves around the critical zones smoothly. This concept works well as long as only few moving objects are in the close vicinity of the robot and the robot has sufficient space to avoid them. But scenes can occur where the robot cannot escape the objects. For example if two objects move in parallel and trap the robot in between them or too many objects are heading for the robot from different sides. To avoid these situations, the predictive handler for moving obstacles will be introduced in the next section. The combined concept is based on the division of duty between a predictive part for the mid-term control and a part for the reactive micro management.

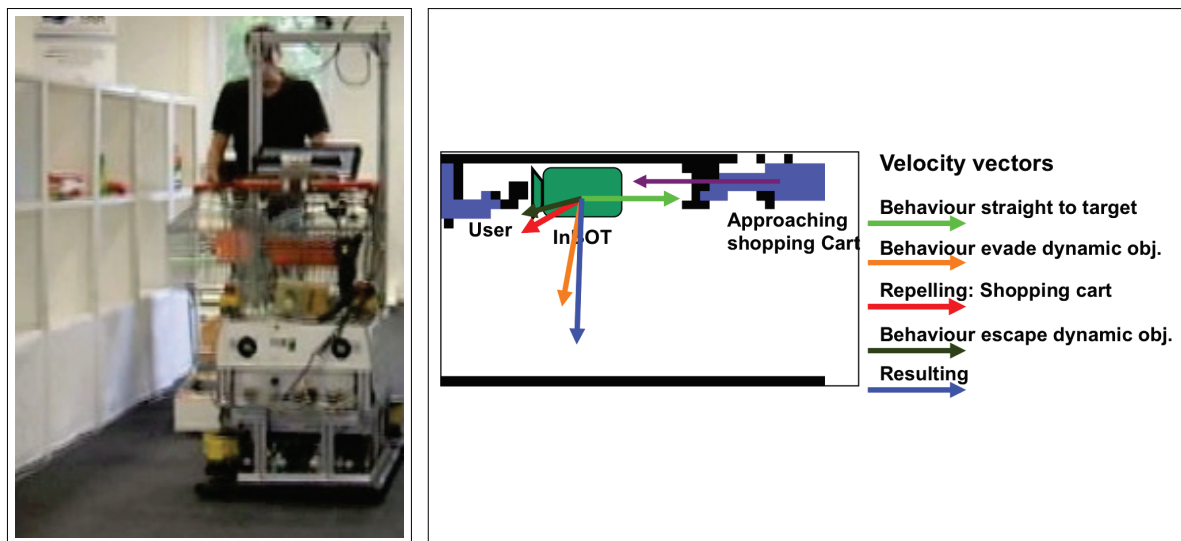


Fig. 5.7.: In a corridor the robot *InBOT* is confronted with a shopping cart approaching from the front. On the right hand side the corresponding occupancy map with the velocity set point vectors from the individual behavior modules is sketched. The next figure (Fig. 5.8) shows *InBOT*'s reaction.

Tests have proven the expected limitations of the local and reactive approaches. Two examples can be seen in Figure 5.9. There are three major kinds of situations: (1) either the repelling vectors force the robot into a trap, (2) the situation can only be solved by withdrawing, or (3) the situation is very confused due to fuzzy information and/or a large number of objects. The first situation takes place for example when a objects moves in parallel to a wall in some distance, and the robot is closer to the wall, or when two objects move in parallel and the robot is in between. The second situation happens when the robot meets an approaching object in a corridor which is too narrow to pass each other. Here one of the two has to withdraw. This is be handled by the *Escape* behavior, but not in a satisfying manner. And in the third case with the large number of objects and the fuzzy information it would be best to avoid the crowd of moving objects altogether.

Obviously the reactive behaviors have to be supported by more anticipatory capabilities. This could be done by more intelligent fusion strategies for the repelling vectors, by modifying the shape of the repelling field depending on the environment and so forth. But this would still be local and reactive methods. Thus, in this work it was decided to provide a plan-based approach to be able to solve situations of all complexities.

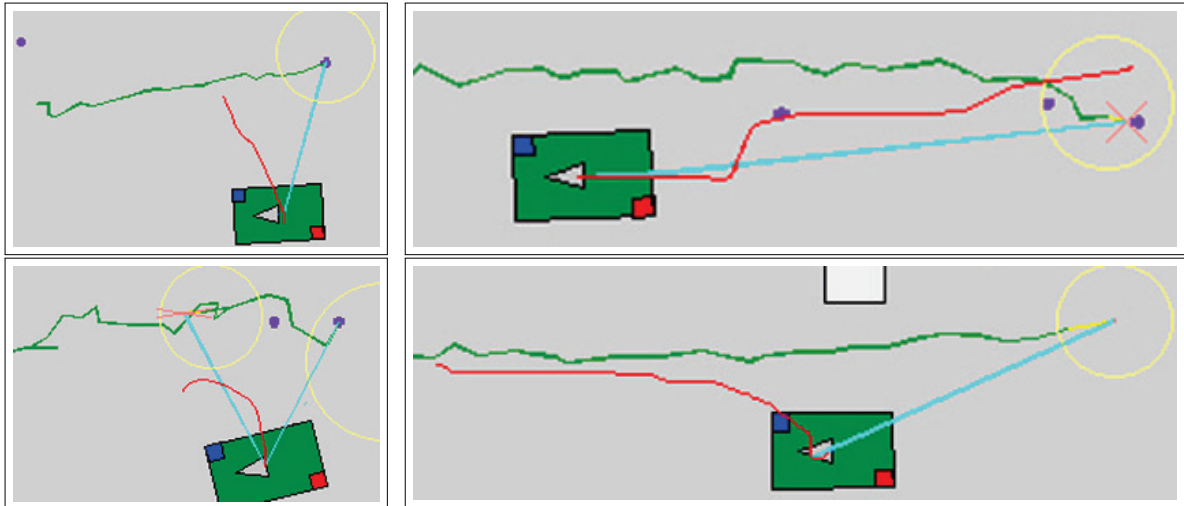


Fig. 5.8.: Resulting evasive motions (the robot is indicated by the green rectangle, the moving objects's track by the green line and the robot's path by the red line).

Left: The standing robot frees the path of an approaching object. In the bottom scene the object was very hard to track resulting in a jittery prediction and thus a not optimal movement of the robot.

Right: Here the robot is moving itself while confronted with an approaching object (top: head to head, bottom: object is overtaking the robot).

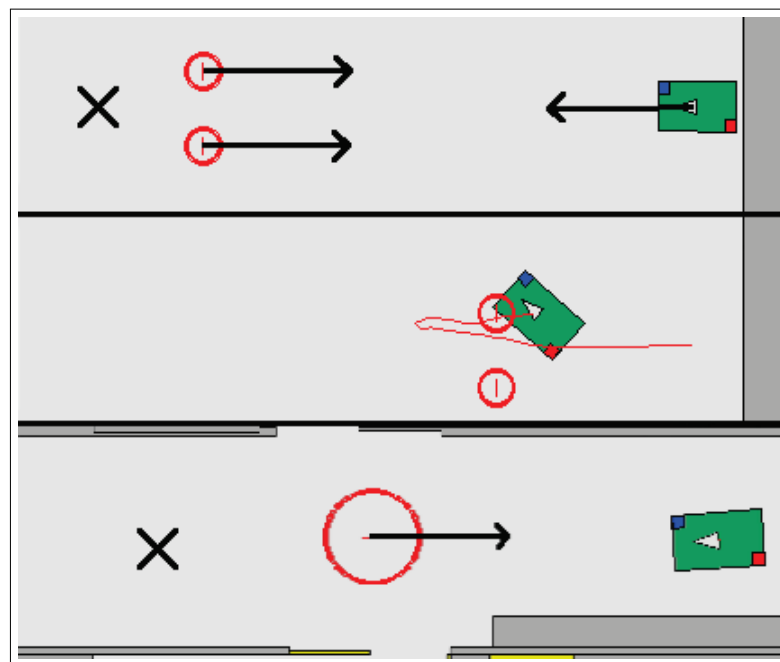


Fig. 5.9.: Sketches showing the limitations of the reactive behaviors.

Top: Two objects (red circles) approach the robot along the indicated arrows. The robot is located in the middle between the two objects. By adding the two virtual force vectors, the robot will be forced to stay between the two objects.

Middle: As expected in the description of the top figure, a collision occurs. The red line illustrating the path of the robot shows that the *Escape* behavior tried to solve the situation, but it was too late as it had to decelerate the robot before accelerating in the opposite direction.

Bottom: Another example which cannot be handled by the reactive behaviors: a large object which fills the complete corridor forces the robot to withdraw. The *Escape* behavior will try to keep a safety distance but the retreat will be in an inefficient manner.

## 5.2. Proactive avoidance of moving objects using spatio-temporal plans

The two presented *reactive behaviors* manage the short term collision avoidance with moving objects with fast reaction times. But they do not perform a thorough analysis of the scene. This done by the planner being introduced here (originating again from [Ste09b]). It generates a spatio-temporal sub-plan in the local environment to reach a given target while taking the predicted movements of all detected moving objects into account. The goal is to basically avoid critical situations which could not be solved by reactive behaviors only (e.g. trapping situations between objects) or to avoid critical areas altogether. The plan consists – according to the requirements of the *tactical layer* – of sparsely distributed sub-targets with time constraints, it is not a trajectory for the robot.

**TB: Plan execution** The execution of the plan is performed by a associated plan execution behavior module which hands new targets down to the *reactive behaviors* one by one – which again guarantee the safe behavior of the robot even if the planning process should fail or provide illegitimate goals. Figure 5.10 shows the implementation of these two modules in MCA2.

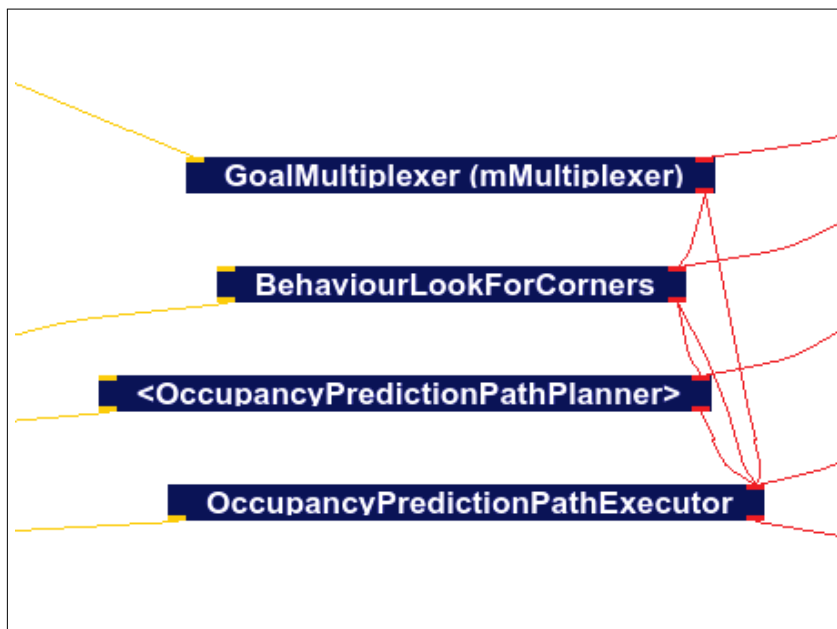


Fig. 5.10.: MCA implementation of the proactive planner: one module generates the plan, a second module inserts the sub-goals of of the plan into to goal-refinement chain of the *tactical layer*.

### 5.2.1. Baseline: Data and pre-processing

The environmental robot-centered 2D occupancy map of the *LWM* (in the case of the robot *InBOT* with a size of  $10 \times 10m$ ) is extended with a time-dimension generating a 3D occupancy map (*Z*-axis for time). Static objects are assumed as time invariant and therefore span the complete *Z*-axis. The position of moving objects is altered according to the movement estimation of the object (see Fig. 5.11). A new grid has to be built for each plan: The time-distance  $t$  between two X-Y layers is set so that it matches the time interval needed by the robot to move one cell along the x axis at default speed ( $t = \frac{cellsize}{v_{def}}$  – with  $v_{def}$  usually 0.5

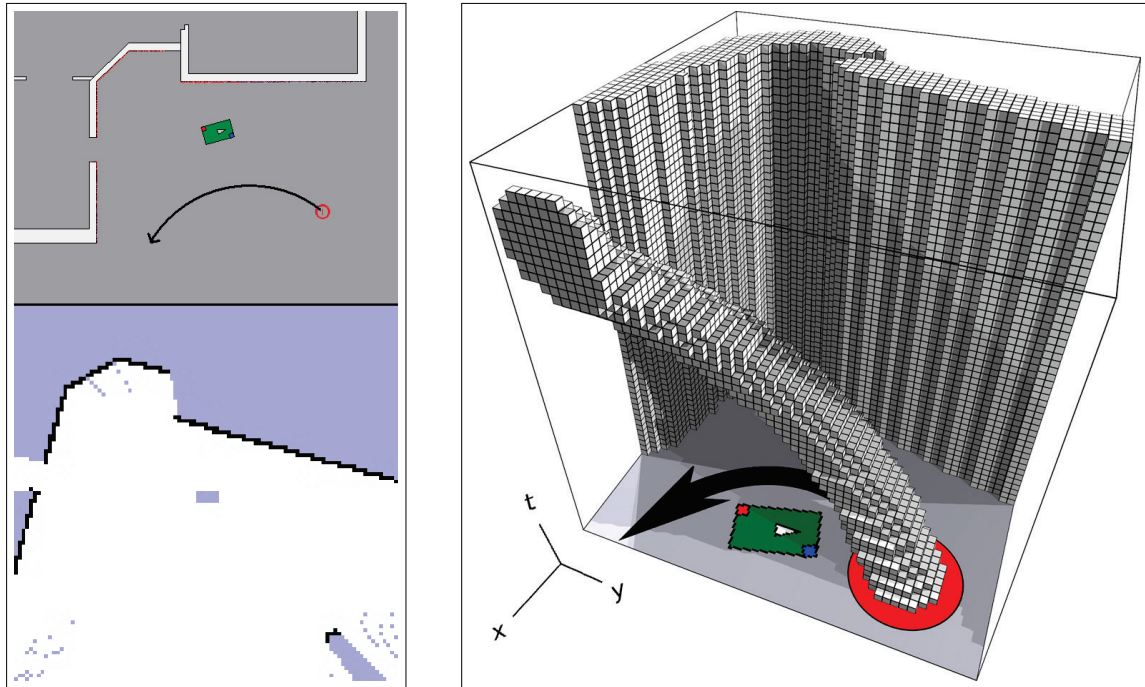


Fig. 5.11.: All three figures show the same scene: a moving object (red circle with arrow) moves in the close vicinity of the robot (green rectangle) in a curved path (indicated by the black arrow).

Top left: The scene viewed in the MCAGUI. The tiny blue and red dots represent the measurements of the laser range finders.

Bottom left: The corresponding occupancy map is shown (the robot is located at the blue rectangle (but larger), the object is represented by the occupied cells (black) in the bottom right corner).

Right: predictive 3D occupancy map with enlarged obstacles ( $50 \times 50 \times 50$  cells, cell size:  $20\text{cm} \times 20\text{cm} \times 0.4\text{sec}$ ). The grey cubes represent the blocked cells – by the static (and thus time-invariant) wall in the background as well as by the moving object indicated by the arc of cubes in the foreground. As the obstacles are enlarged, the illustration of the robot is too large here – the robot actually has the size of one cell.

to  $1\text{ m/s}$ ). The height (time) of the 3D occupancy grid limits the length of the path that can be found. The resolution of the basic occupancy grid is reduced and moving obstacles as well as special objects (the user, other low-priority robots (see chapter on multi-robot behaviors: Chap. 7)) are deleted from this occupancy grid. The static obstacles are enlarged (the size of the robot can now be assumed as point like) and added to the 3D grid. Additionally, for each time-step an enlarged obstacle is entered at the predicted positions of the moving obstacles. The resulting predictive 3D obstacle map is shown on the right-hand side of Fig. 5.11. It was decided to generate a new plan each time instead of altering the old one, because of the drastic changes in the local map due to the changing field of view and perspective when the robot moves, the dependency of the robot's actual velocity and finally the fuzzy prediction of the moving objects' behavior which suddenly can be subject to major changes.

### 5.2.2. Spatio-temporal calculation of safe path using an $A^*$ algorithm

A starting cell and the goal coordinates have to be given to calculate a path around the moving obstacles in the predictive 3D occupancy grid. The robot is positioned at the middle of the lowest time-layer of the grid – the start cell. As it is not important at exactly which point in time the robot reaches the goal, all cells that share the X-Y-coordinates of the goal point are possible goal cells for the  $A^*$  search. If the goal point is

located outside of the occupancy map a substitute goal point located on the grid's border is chosen. Figure 5.12 illustrates this setup.

The  $A^*$  algorithm is forced to take a time step for each X/Y movement, therefore a cell has 9 neighbors which are located in the succeeding time layer  $t' = t + 1$ . Occupied cells in the 3D obstacle grid are obviously off-limits.

The heuristic function used for the  $A^*$  search is the straight-line distance to the goal in an X-Y layer ( $\sqrt{\Delta x^2 + \Delta y^2}$ ) multiplied with the factor  $\sqrt{2}$  to estimate the needed time to get to the goal (this is equivalent to the straight-line distance in the 3D occupancy grid). This factor is a correct estimation if the  $A^*$  search can find a free way to the goal without having to wait or having to take a detour around an obstacle.

The search will be aborted if the current cell in the  $A^*$  search is located in the top X-Y layer  $t = t_{max}$ .

If the  $A^*$  search does not find a path to the goal there is no free path to the goal in the given time constraint and the search has to be restarted on the next new occupancy grid data. In the meantime the robot relies on the *reactive behaviors* to avoid collisions.

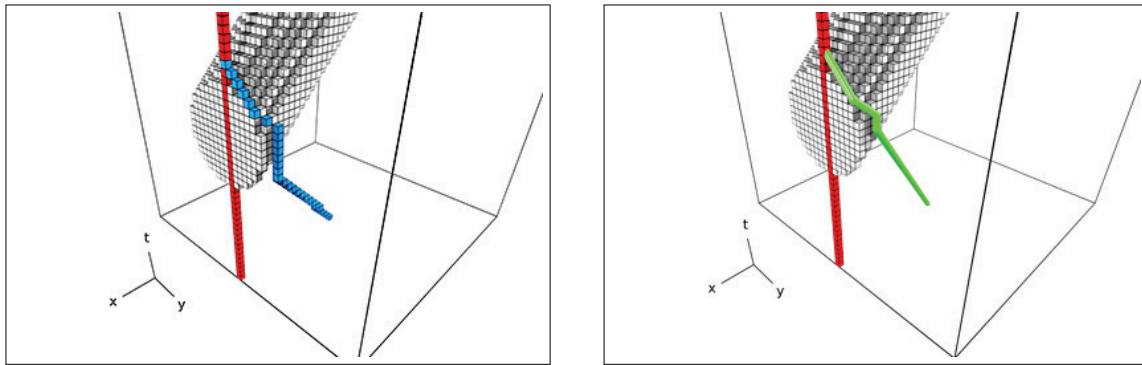


Fig. 5.12.: The robot plans a path to a location which will be crossed by a moving object. So the robot has to reach the goal after the moving object has moved past it. The robot is located at the center of the grid (in the center of the X-Y-Plane, at the bottom regarding the time dimension). The goal is marked with the red boxes on the left hand side of the grid – obviously, it is suitable to reach the goal at any point of time, so the goal location is marked as goal throughout the complete time span.

Left: The blue boxes show the path planned by the  $A^*$ .

Right: The green line is the optimized path. It only consists of the few kinks of the green line (accompanied by a time constraint).

### 5.2.3. Optimization of path

The path calculated by the  $A^*$  consists of a list of sub-goal points. If the sub-goals are located too close to each other the flexibility of the *Behavior-Based Control* is impaired. If there are no obstacles, the path still consists of many sub-goals in a straight line even if only the goal point itself would be sufficient to provide a collision-free path.

Therefore, the path is simplified to contain only the needed points to drive around the obstacles by repeatedly removing sub-goals and then testing for collisions with occupied cells (see Fig. 5.12). If the resulting path is collision-free the removed sub-goals are removed permanently.

### 5.2.4. Utilization of the *Behavior-Based Control*

Once a path is found and simplified, its sub-goals are given to the *Behavior-Based Control* as goal points. The *Behavior-Based Control* moves the robot to these points successively while utilizing the full range of *reactive behaviors*.

### 5.2.5. Experimental results using the planner

The described algorithm was tested thoroughly. An easy example is displayed in Fig.5.13. A moving object approaches from the direction the robot intends to move in. After detecting the object, a plan is generated to move around the object. A more challenging scene is depicted in Fig. 5.14. This time, there is not sufficient space to move around the object. Therefore, the plan leads the robot back around a corner to let the object pass before continuing with moving towards the target.

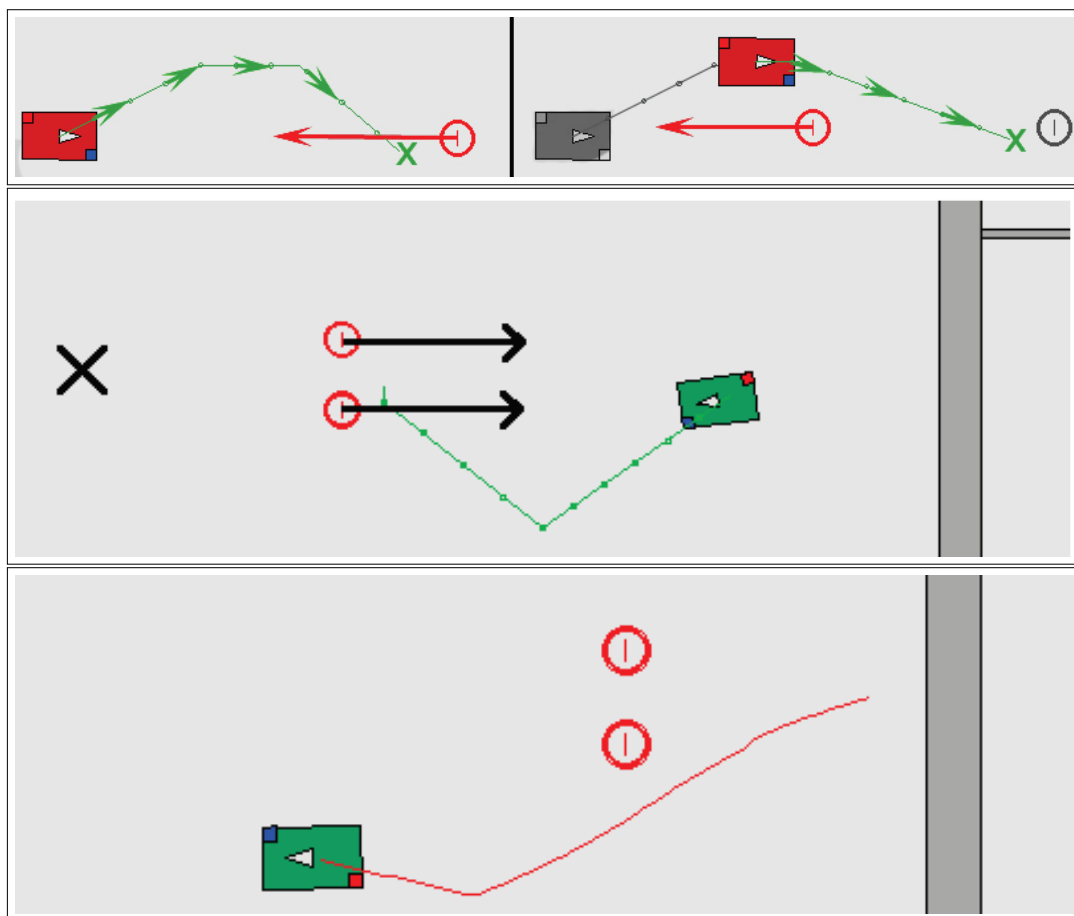


Fig. 5.13.: Top: The robot's target is the green "X" at the right side of the scene. A moving object approaches from this direction (red circle with arrow). Therefore a plan of sub-targets is generated that leads the robot around it. Middle and bottom: The scene from Figure 5.9 is shown again. Here the *reactive behaviors* failed. As indicated by the plan (green line with arrows) and the path taken by the robot (red line), the planner is able to solve this situation.

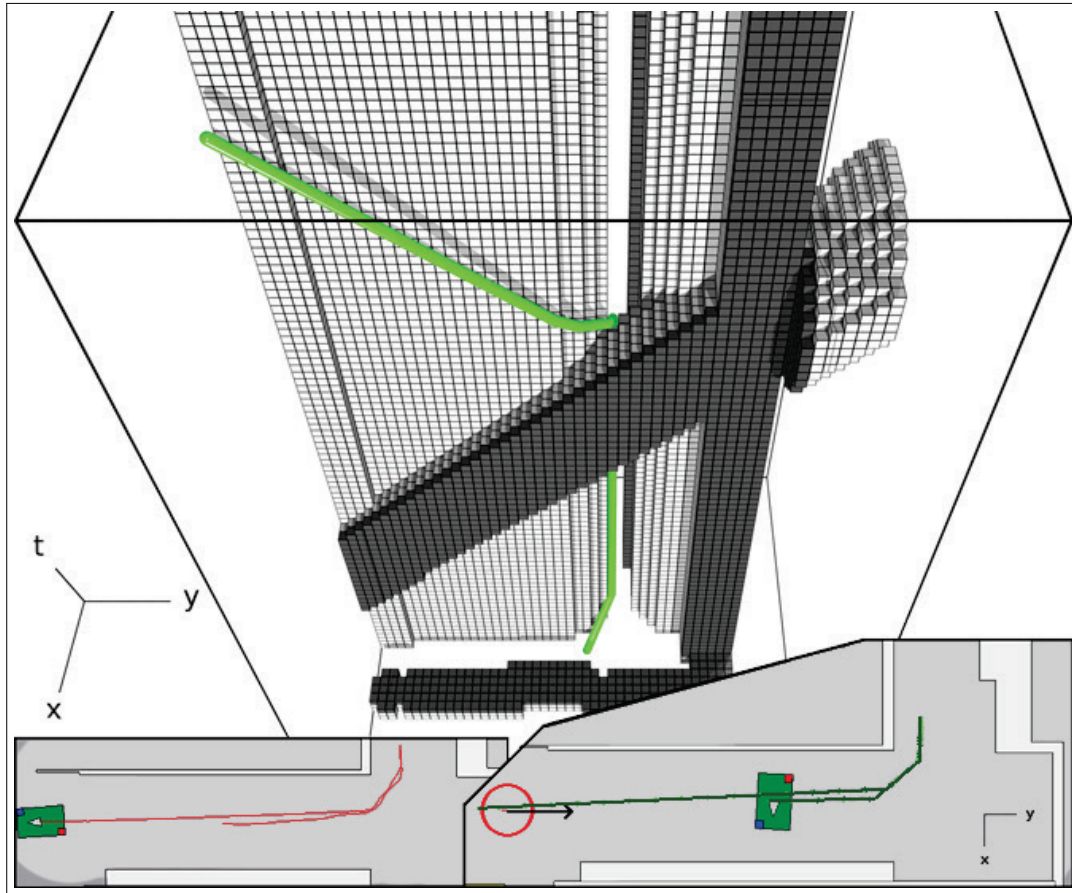


Fig. 5.14.: Here again a scene is taken up from Figure 5.9 where the *reactive behaviors* failed. The robot's target is at the left hand side end of the corridor. A large moving object approaches from this direction (red circle with arrow). Therefore, a plan of sub-targets is generated that leads the robot back around the corner, lets it wait there until the object passed and finally leads to the target again. Top: plan in a cut open 3D view. Bottom/right: plan in 2D view in MCAGUI. Bottom/left: actual path taken by the robot.

### 5.3. Experiments and tests

In this section some of the experiments performed to evaluate the concept will be presented. When not stated otherwise, all experiments were performed with the real robot in the real environment.

#### 5.3.1. Comparison of components

Figure 5.15 shows the effect of the different behaviors for obstacle avoidance. Using only the behaviors for the avoidance of static obstacles a collision occurs as long as the robot is not significantly faster than the object. The more specialized the behaviors used are the earlier the robot starts the evasive motion and the more efficient and safe becomes the path.

#### 5.3.2. Stress test

A stress test was performed in a partially simulated setup: Three objects are moving back and forth on straight paths always crossing the robot's target point. The robot's tracker for moving objects was manipulated to use the simulate objects to be able to test the collision avoidance system independently from the



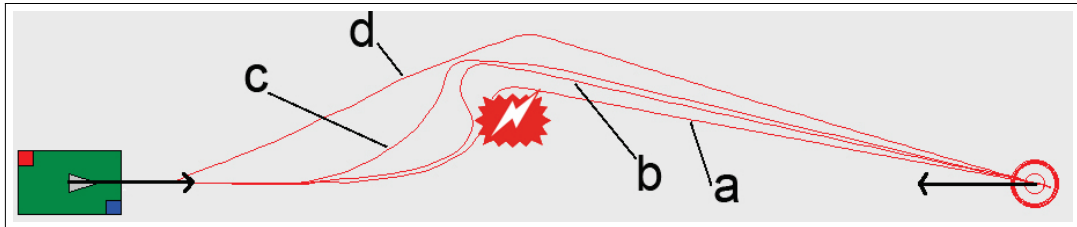


Fig. 5.15.: The individual behaviors for the obstacle avoidance in comparison. The robot moves from the left to the right, the red lines shows its path. (a): *AvoidObstacles* (static), (b): *Escape* behavior, (c): *Evade* behavior, (d): Proactive planner. On path (a) most of the time a collision occurs (indicated by the flash symbol), depending on the proportion of the robot's velocity compared to the object's velocity. The *Escape* behavior is most of the times able to avoid a collision (again depending of the velocity difference of robot and object), but the path is not efficient. The *Evade* behavior avoids the approaching object robustly and with a suitable path. The planner's path is the best, but there is a delay at the beginning due to the slower reaction time. This did not pose a problem here as the object was clearly visible up front, but it could be a critical issue.

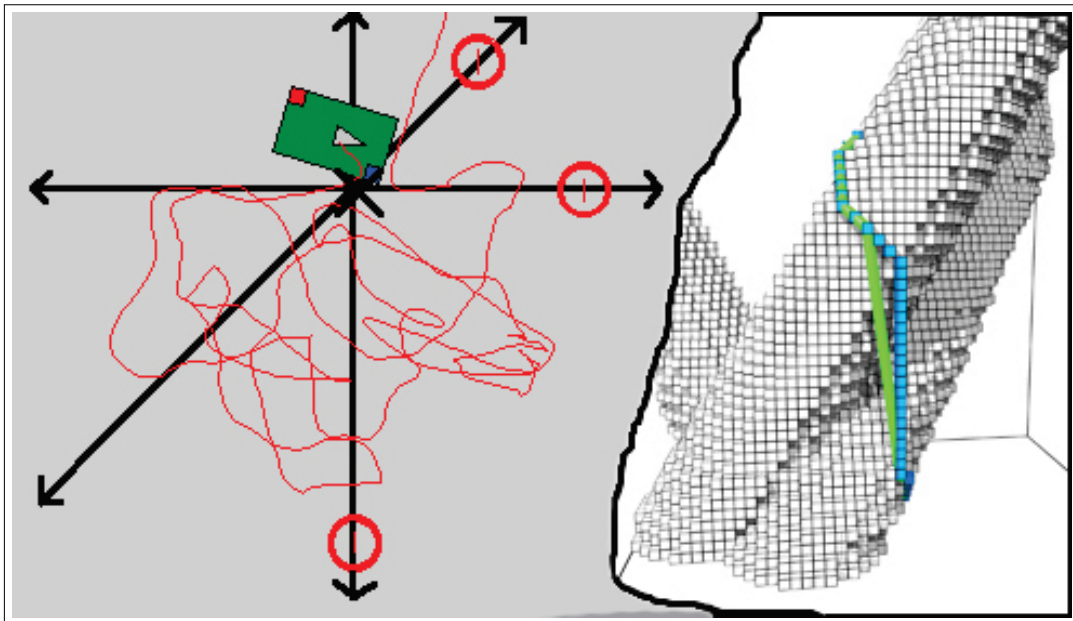


Fig. 5.16.: Stress test: The robot is ordered to move to the X in the center while three objects move back and forth on paths crossing just this point (indicated by the red circles moving along the black arrows). During a two-minute test run there were 50 possible collisions. On the right hand side an exemplary extract from the planner with the calculated and optimized path is shown. The results are provided in Table 5.1.

## 5. Avoiding Collisions with Moving Objects

quality of the tracker. The (real) robot has to continuously dodge the objects. Figure 5.16 illustrates the setup and the path the robot has actually taken during a two-minute test run.

Table 5.1 shows the results of the stress test for the individual behaviors. Using only the behaviors for the avoidance of static obstacles 34 collisions occurred – which is less than expected. Using the *reactive behaviors* *Escape* and *Evade* the number of collisions goes down to a single one. When additionally using the planner the distance to the closest object is significantly higher meaning a safer path. The single collision that occurred in both cases was unavoidable because the (simulated) objects turn around – beyond the laws of inertia – without having to (de-) accelerate. If the robot is located close to a turning point, the robot cannot accelerate away quickly enough.

Tab. 5.1.: Results of stress test: 50 passes during two minutes (see Fig. 5.16)

Method used	Collisions	Average distance to closest object
Static avoidance	34	146 mm
Escape and Evade	1	665 mm
Above + Planner	1	1041 mm

A second stress test was performed in simulation to prove the deterministic manner of the behavior group for avoiding moving objects: the robot was ordered to travel back and forth between two locations 50 times, while being impaired by three moving objects. The overall situation is always roughly the same. Figure 5.17 shows that the path taken by the robot is very similar in all cases.

Tab. 5.2.: Overall success rate: 100 trial runs each

Tracking information	Collisions	Risky	Safe
Using tracker	13	26	61
Using exact position and velocity	3	10	87

### 5.3.3. Overall success rate

Table 5.2 shows the overall success rate over various experiments. The robot *InBOT* (the real robot) was approached 100 times by (real) shopping carts while performing various tasks or just standing still (Figure 5.18 shows some pictures). The ordinary shopping carts were steered by people who moved to a given goal chosen to provoke a collision while completely ignoring the robot, but some even rushed directly onto the robot, trying to test it – which was not intended but provided valuable experiences.

In the first 100 tests the on-board object tracker based on two laser range finders was used, for the second 100 passes the exact positions of the shopping carts were provided by using the system *ETrolley* which has its own self-localization equipment. The amount of safe runs shows that the collision avoidance concept performs well but tracker has to be further improved or a better one has to be integrated. The majority of collisions and risky runs (slight touching not hard enough to activate the robot's bumper) was caused when the cart was identified too late and the robot did not have sufficient time to accelerate away. There were 3 collisions and 10 risky situations in the run with the exact cart positions in which the drivers of the cart did not let *InBOT* a chance to avoid them properly. The reason is the acceleration and velocity limitation of  $1m/s$  set for the robot, which was significantly lower than the speed of some of the cart drivers. Additionally,

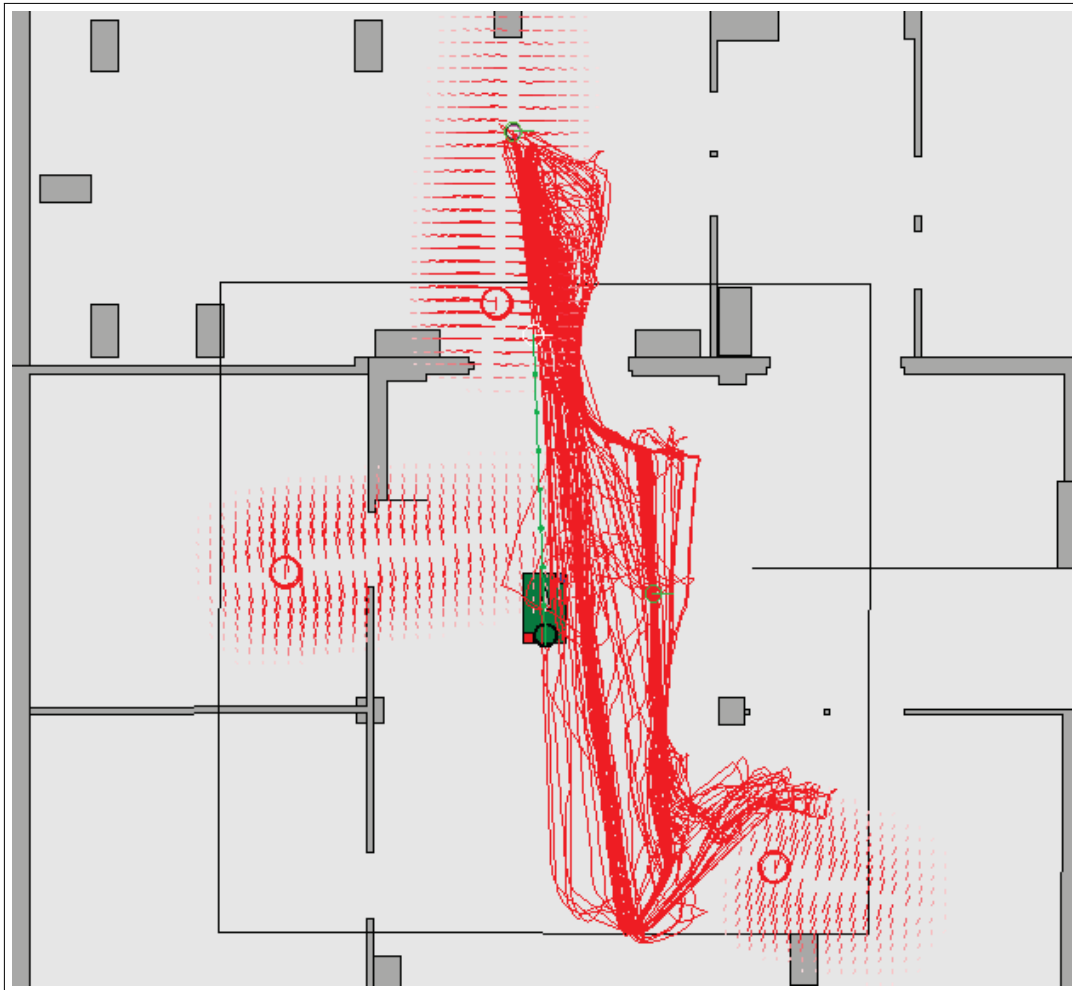


Fig. 5.17.: Second stress test to prove the deterministic manner of the behaviors: The robot is ordered to move back and forth 50 times between two locations (top and bottom turning point) while being impaired by three crossing moving objects. This setup gives always similar – but not exactly identical – scenes. This figure shows that the path actually taken by the robot is always very similar, in most cases almost identical.

some cart drivers forced the robot against a wall were *InBOT* fell to a dead stop shortly before hitting the wall due to the safety behaviors – and then was hit by the cart.

Keeping in mind that the intention is to avoid accidental collisions, the robot completely fulfilled the expectations. Finally, it shall be noted here that in all cases of collisions *InBOT* was hit by the approaching object due to insufficient dodging – *InBOT* itself did never run into any objects or into the path of an approaching object. Therefore safety from the robot's point of view was always granted.

#### 5.3.4. System in application

The last test to be presented here includes some scenes from a shopping experiment – involving all available functionalities – where the robot is hindered in executing given tasks by moving obstacles (e.g. other shoppers with shopping carts). Fig. 5.19 shows a single shopping run in a small laboratory supermarket setting. The same experiment will be presented as well in the next two chapter focusing on the user interaction and the multi-robot coordination.



Fig. 5.18.: Photos showing some of the performed tests using a ordinary shopping cart and *InBOT*'s on-board object tracker. Others have been done using *ETrolley* to gain precise and robust localization information on the moving object.

The user is asked to pick up five products while other shoppers are moving around him with their carts. First the robot guides the user to the first product and simultaneously avoids the moving obstacle (A), later while operating in the *Following Mode* the robot keeps a safe distance from obstacles (B) and (C). In all cases the robot did not move into other objects and did not collide with its user (as described in Chapter 3 the presented behaviors are merged with the behaviors for handling static obstacles and the safety module counterchecks all movement commands in the end).

### 5.4. Discussion

Some methods for avoiding moving objects proposed in the literature only steer the velocity of the robot on the path planned around the static obstacles, which is only sufficient for moving obstacles crossing the path, not for ones moving along the same or a similar path. Others are directly coupled to the path finding algorithm. This is not sufficient as well because in a dynamic environment the robot can also be hit by moving objects while standing still. In general, the advantage of the deliberative planners compared to the reactive methods is obvious – especially when state-of-the-art hardware renders the time delay for the computation insignificant. But the scenario of this thesis lacks too much information (limited field of view, no exact and reliable map, sometimes no target location is available e.g. in *Visual Servoing Mode* or *Manual Steering Mode*) to be able to rely always on a planner-based approach only.

Therefore, in this thesis a combination of a planner with reactive behaviors is used. Again a three-leveled approach is proposed: First, a reflex moves the robot directly away from mobile obstacles which came too close or suddenly started moving, enabling the robot to regain a safety distance. This reflex is kept extremely simple, this way it is able to dodge moving objects even before a prediction of the movement direction can be made. Second, a reactive behavior lets the robot move out of the predicted path of an approaching object. Third, to solve complex situations, the behavior-based components are topped by a spatio-temporal planner which generates a safe and efficient long-term path.



Fig. 5.19.: Final test for avoiding moving objects – a shopping run involving all available functionalities: the user (tracked by the blue line) starts together with the robot (grey rectangle, red line) in the lower right corner. The robot guides the user to the first product (1) while avoiding the moving shopping cart (A) (indicated by the yellow dotted line): the strong bend to the left at the yellow flash symbol. After following the user to the next products (2) and (3), the robot continues to product (4) and avoids the crossing cart (B) (orange dotted line) by slowing down and letting it pass. The robot continues by aligning itself behind the cart (orange flashes). While following to the final product (5) the robot is cut by cart (C) (red dotted line). Again, the robot lets the cart pass (red flash). (Robot's path red, critical areas: flash symbols, the user – who is continuously impairing the robot being another close-by moving obstacle – is tracked by environmental cameras resulting in the chaotic blue line. The bad quality of the user's track comes from continuous occlusions by other objects (shelves, robot, carts, other shoppers, etc).

## 5. Avoiding Collisions with Moving Objects

---

In several tests the concept has proven its performance. The system was designed to avoid accidental collisions and performs well under this assumption. If someone should really try to hit the robot he will succeed due to the velocity and acceleration limitations applied on a robot operating in a populated environment.

A crucial challenge in avoiding moving objects is the robust and timely detection and tracking of the objects. In this thesis two components were utilized to provide the necessary data: First this were two planar laser scanners mounted on the robot at feet height. This is not sufficient for robustly tracking persons as occlusions are frequent and the field of view is very limited. Additionally tracking a group of walking persons is prone to errors because the persons lift the feet above the measurement plane of the sensor when walking. Therefore the second component used two planar laser range finders mounted in the environment at chest height whose data has been merged with the data from environmental cameras. The drawback on this system has been the additional time delay resulting from the transfer of the data to the robot using WiFi.

## 6. User Interaction

After the navigation system has been described, this chapter approaches the second major challenge: the human-robot interaction (HRI) capabilities of the control system. A special focus is on a challenge common for cooperative actions of robot and user: sharing the control between robot and user smoothly. This way the robot can assist the user with the implemented abilities without putting the user out of control. Preluding, the interactive capabilities, modalities for the users' input, and the *modes of operation* of *InBOT* will be discussed.

As the shopping assistance robot shall serve its user, interaction with the user is an important topic. Having a powerful navigation system alone may serve the robot well, but not the user. The user needs to have access to the functionalities in an intuitive way – he wants to do shopping and not to control a robot – and the robot must be able to fulfill tasks in cooperation with the user. Especially handicapped users could benefit from the provided assistance functionalities: visually impaired customers can command the robot by speech and let themselves be guided around obstacles and directly to the desired products. Weak or elderly people do not have to push the robot themselves. But especially when cooperating with these groups of people an robust HRI system is even more important.



Fig. 6.1.: *InBOT* is guiding a user

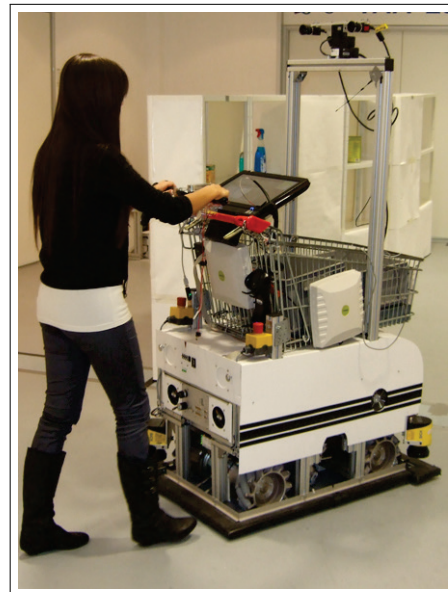


Fig. 6.2.: *InBOT* is controlled by the *force sensitive handle*

In contrast to tele-operated or autonomous robots, in the setting of this thesis the user is always present, even impairing the robot's motion. Thus, this chapter shall in particular focus on a challenge common for

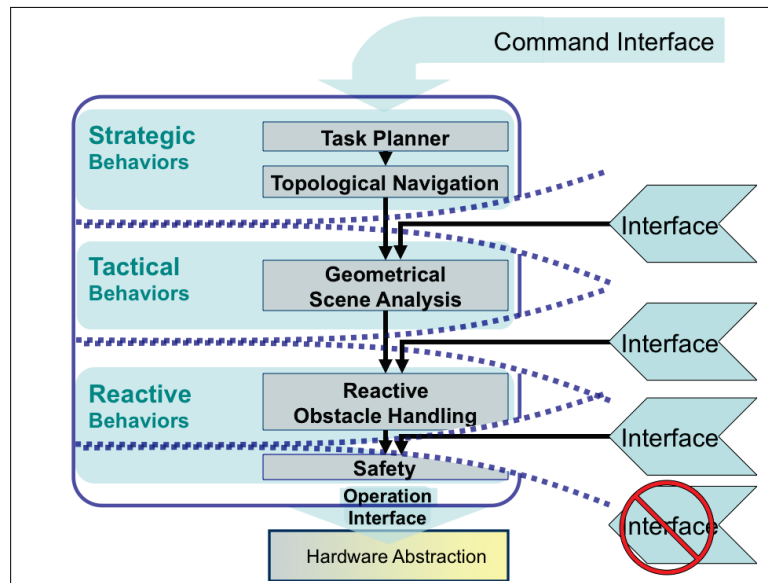


Fig. 6.3.: Inserting control data into the architecture using interfaces between layers. Insertion of user input between the safety behaviors and the *Hardware Abstraction Layer* is not allowed to ensure that the safety behaviors are always in charge.

all cooperative actions of robot and user: the trading and smoothly sharing of control. As baseline for the interaction the modalities available to the user for controlling the robot are identified. They inform the robot about what the user actually wants to do. A second foundation for the interaction are the *modes of operation* (two examples can be seen in Fig. 6.1 and Fig. 6.2) which define the current setting for the cooperative task execution. The chapter will be concluded by showing the application of the concept of *control sharing* and *Trading* in some experiments with the robot *InBOT* and human users.

As discussed in Chapter 3 “The Hybrid Control Architecture”, the concept of *control sharing* is not implemented by associated behaviors. It emerges from a multitude of components and their interplay as defined by the architecture, such as but not limited to the integrated multimodal *communication layers*, the *force sensitive handle*, the user tracking, or finally the adaptive behaviors. The HRI system spans the complete control architecture and thus is present on all levels of abstraction. Very important for *control sharing* is the control data flow as it defines how the user can influence the robot’s behavior. The control architecture defines user-input orthogonally to the usual control data flow by providing interfaces and fusion behaviors on top of each of the layers (see again the sketch shown in Fig. 6.3).

**Scope of this chapter:** Due to the enormous wide range of the field of HRI, the content implemented in the course of this thesis will be limited to the components and concepts relevant for the application of shopping assistance in the supermarket, for example implementing the guiding and following behaviors, giving orders to the robot, receiving feedback from the robot, and sharing the control during task execution. The HRI shall focus on the human acting as the customer to whom the robot is attached, other humans are regarded only as moving obstacles in most of the cases. Humans acting as operators or supermarket staff will be omitted here completely. They can either control the robot using the same modalities as the customers do or exert direct control using the *MCAGUI*. Further concepts like interpreting the humans’



intentions, learning, and actual collaboration or even human-robot teams will not be considered as these fields are complete topics for research on their own.

**Content and organization of the chapter:** Following after this introduction, in Section 6.1 the five *modes of operation* (autonomous, guiding, following, servoing, and manual steering) are introduced, which enable the user to utilize the robot according to his wishes. In the subsequent section (Sec. 6.2) the means and modalities are described by which the user is able to influence the robot such as the touch screen interface or the *force sensitive handle*. The next section describes that mode transitions are performed based on the modalities chosen by the user. Section 6.4 describes the interaction of robot and user regarding the chosen modalities like the obstacle avoidance assistant or the *Force Commands* during *Manual Steering Mode*. Additionally, tactical behaviors dedicated to HRI are introduced. The last but one section (Sec. 6.6) elaborates on the concept of *control sharing*, how it emerges from the components described in the preceding sections, and its influence on the control. And finally the last section concludes the chapter by describing experiments and the results from surveys conducted with the robot's users after the experiments.

## 6.1. Five *modes of operation*

Different users prefer to use the shopping assistant robot in different ways. Some users want to be completely in charge, others just want to be guided through the shop. These varying interest of the users are reflected by introducing the *modes of operation*. Each mode represents a unique way of utilizing the robot. Even though the basic components used by the robot's control system (such as obstacle avoidance) are identical, the experience of using the robot differs significantly.

As was defined as requirement in the introduction (Chapter 1.4.1 "*Modes of operation* for user interaction (R2.3)") the utilization of the robot's capabilities are grouped into five *modes of operation* to suit the wishes of the individual groups of users. Ordered by their degree of coupling between user and robot these are:

- *Manual Steering Mode*
- *Servoing Mode*
- *Following Mode*
- *Guiding Mode*
- *Autonomous Mode*

The first mode with the closest coupling between user and robot is the *Manual Steering Mode*. Here *InBOT* can be steered just as an ordinary shopping cart by the *force sensitive handle*. The control system supports the user with several assistance features such as an navigation and obstacle avoidance assistant, local *Force Commands* (e.g. "Park on side", "Turn to support loading"), and obviously the full motor power of the robot.

The second mode is the *Servoing Mode*. In this mode the robot imitates the movements of the user (or any other given target) while avoiding collisions. The purpose of this mode is to accompany the user during local movements – e.g. looking for an individual product in a shelf – without impairing him. Obviously, information on the users motion are necessary as data input. These are provided by the *LWM* and can be acquired for example by the *intelligent environment* (see Appendix C.3 "*The intelligent environment*") or

Tab. 6.1.: Dependencies of modes and data sources.

Data requested for	<i>Autonomous Mode</i>	<i>Guiding Mode</i>	<i>Following Mode</i>	<i>Servoing Mode</i>	<i>Manual Steering Mode</i>
Command for CL from	—————	high level UI	—————	automatic	handle
Task input to <i>strategic layer</i>	<i>communication layer</i>		/	/	/
Target input to <i>tactical layer</i>	— <i>strategic layer</i> —	user's position		/	/
Velocity input to <i>reactive layer</i>	—————	<i>tactical layer</i> —	—————	UM	handle
Velocity output to HAL	AO	—— AO and user's position ——		AO, UM	AO, handle
Obstacle avoidance and safety	X	X	X	X	X

CL: *communication layer*, UI: user interface, AO: *Avoid Obstacles* beh., UM: user's motion, X: active

the *user tracking with onboard sensors* (see Appendix C.2 “Detecting and tracking the user using onboard sensors” and [63], [59]) but other means can be utilized likewise.

The third mode is the *Following Mode*. Here the robot follows the user (or any other given target) wherever he goes in a well-defined distance. Obviously, again information on the user's position and motion are necessary.

The fourth mode is the *Guiding Mode*. In this mode the robot guides the user (or again any other target object) to a desired goal location or product or even along a whole shopping list step by step while continuously adapting to the user's pace. When starting a motion in this mode the robot starts slowly: it uses only half the usual acceleration and moves with 0.25m/s maximum velocity for the first meter of distance. This way it shall avoid to startle the user with a sudden movement as well as giving the user enough time to cancel the action, even by using the touch screen which is difficult to use during motion. Again for this mode, information on the user's motion is necessary.

The last mode is the *Autonomous Mode* with the weakest coupling between user and robot. Here *InBOT* acts only if explicitly commanded by the user, and once commanded the robot performs the task independently.

**Influence of the modes:** The active *mode of operation* has two kinds of influences on the control system: firstly a mode transition triggers a module which sets the motivation or inhibition of certain behavior modules such as the adaptive behaviors (see Chap. 4.4.2 “RB: Behaviors for adapting the velocity to the task and the user”). And secondly the mode organizes the assignment of target locations. For example during *Guiding Mode* the target location is taken from the *strategic layer* and the velocity adaptation behavior is active while during *Following Mode* the target location is the user's position and the distance adaptation behavior is active. During *Manual Steering Mode* no target location is used at all, but the velocity set-point vector used in the *reactive layer* is directly acquired from the *force sensitive handle* or any other force input device – see Table 6.1 for an overview.

## 6.2. Modalities for interaction and the multimodal user interface

There are several ways to interact with or to command a *service robot* from which a user would like to chose. The choice is often not made consciously – the user does not want to have to reason on how to command the robot but simply pick the most convenient method. Thus, a *service robot* should offer a wide range of methods and modalities and switching between them should be easy and intuitive. Hence, as many different options as possible should be offered to the user, hoping that always the most convenient one is available.

In the following the various modalities are summarized which are available for the user to exert control over the robot directly or to indirectly command the robot. The modalities are connected to different layers of the control architecture and provide for different kinds of information. For some modalities alternative implementations were integrated to improve the evaluation of the overall system.

- Multimodal user interface of the *communication layer*. It receives and pre-processes the user’s high-level commands. Two versions were used: the *CR-UI* (see [87] and Appendix C.5) implemented by TU Vienna and the *InBOT-UI* (see Appendix C.4) developed in the course of this thesis (see Figure 6.4). Both provide the same modalities:
  - Touch screen based graphical user interface with shopping list management and buttons for the individual commands.
  - Speech recognizer for the individual commands such as “guide me to <product>”, “guide me to next product” or just “follow me” or “come here” (*CR-UI* only).
  - Bar code scanner for scanning products.
- User tracking: provides position and motion of the robot’s user. Two versions were used: the *user tracking with onboard sensors* (see Appendix C.2 “Detecting and tracking the user using onboard sensors” and [63], [59]) developed at LAAS CNRS and the *intelligent environment* (see Appendix C.3 “The *intelligent environment*”) developed in the course of this thesis. By this modality the user does not exert control intentionally but passively by his mere presence or more precisely by his position and motion in relation to the robot. This modality is crucial when operating in the *Guiding Mode* or *Following Mode*.
- Gesture command recognition for commands such as “come here”, “go there” or “follow me” (only in the integrated version of the user tracking developed at LAAS CNRS: *user tracking with onboard sensors*)
- The *force sensitive handle* (see Fig. 6.4 and Appendix B.6) enables the user to directly provide a velocity set-point vector the be processed by the *reactive layer*. The user can steer the robot manually (including some assistance functionalities and *Force Commands*).
- *Force Commands* for local maneuvers such as “park on side” or “turn around to support loading” (described in the Section 6.4.1)

Finally integrated the modalities provide “the eyes and ears” of the robot regarding the user to facilitate the interaction with the user.

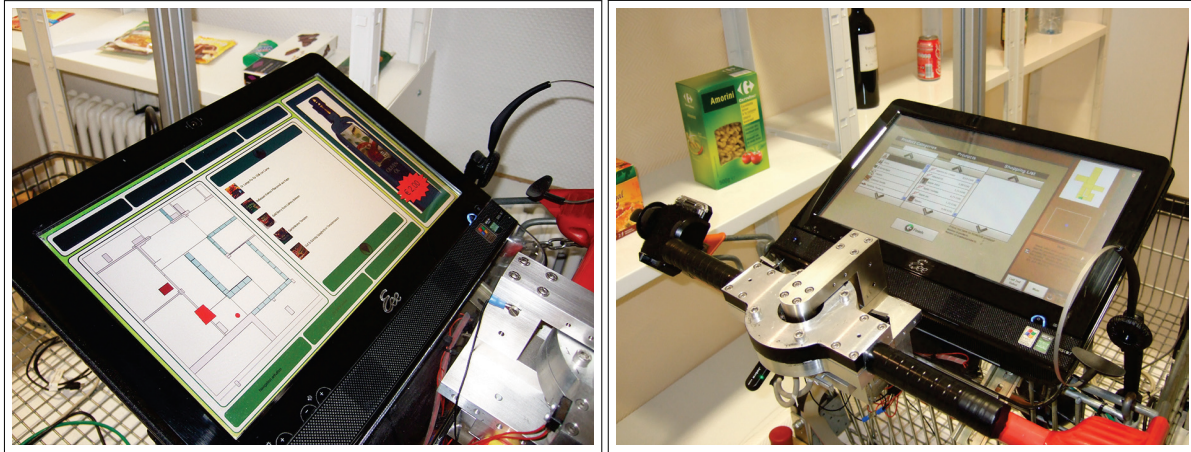


Fig. 6.4.: The user interface of *InBOT*: *force sensitive handle*, bar code scanner, headset, and the touch screen-based GUI. Left the graphical user interface which was developed for *InBOT* – the *InBOT-UI* (see Appendix C.4), right the alternatively integrated *CR-UI* (see [87] and Appendix C.5), both acting as the *communication layer*.

As indicated, some of the components implementing the modalities were developed by other groups. It was aimed at having always two different methods at hand to improve the evaluation of the overall system later on.

### 6.3. Mode transitions

Many users prefer one of the *modes of operation*, but different situations enforce changing the mode appropriately. But for the user it would be inconvenient to have to think about the best suitable mode – or about *modes of operation* in general. In contrast, the transition between the modes shall be performed automatically based on the task given by the user and the modality used for giving that task.

All four modes utilize a subset of the described modalities. This is illustrated in Tab. 6.2. If the user gives a command via a modality that is not coupled with the mode the robot currently operates in, a mode transition is performed. For example if the robot operates in the *Manual Steering Mode* and a voice command is received the robot automatically switches to a mode which is coupled with voice commands. This could for example be the *Following Mode* if the command was “follow me” or the *Guiding Mode* if the command was “guide me to <name of product>”. While usually an explicit command of the user is necessary, *InBOT* automatically switches to the *Manual Steering Mode* as soon as a force is applied to the handle, to idle when the handle is released, and to autonomous when a *Force Command* is detected. The *Servoing Mode* can also be activated automatically after reaching a goal while guiding the user.

The user position tracking does not transmit direct commands to the robot. It passively commands the robot to accelerate or decelerate and therefore does usually not result in a mode transition. An exception takes place if the user is lost. Then a transition to the *Idle Mode* is performed. The usual way for the robot to switch to the *Idle Mode* is the completion or the cancellation of the active task.

The problem of oscillating between modes does not appear here because mode transitions are always triggered by sparse events such as user commands, reaching of a goal, grabbing the handle, etc.

Tab. 6.2.: Dependencies of modes and modalities of interaction.

Mode	Force sens. device	User position	Gesture recogn.	Voice command	Touch screen cmd.	Voice out	Screen out
Manual Steering	X	-	-	-	-	X	X
Following	-	X	X	X	X	X	X
Guiding	-	X	X	X	X	X	X
Autonomous	X*	-	X	X	X	X	X
Idle	-	X <sup>+</sup>	-	-	-	X	X

\*: In the *Autonomous Mode* the handle is used to give force-based commands only, not to steer the robot.

+: If the user is lost the robot switches to the *Idle Mode*.

## 6.4. Interaction regarding modalities

After having introduced the modes and modalities, this section presents the individual methods of interacting with the robot, ordered by the modalities used for the interaction. The main groups are: force input, observation of the user, touch screen input, and voice input.

### 6.4.1. Interaction based on force input by the user

The first and most intuitive method of controlling a *service robot* in a shopping scenario is probably the interaction based on physical contact by using a *force sensitive handle* (see Fig. 6.5 and Appendix B.6 for construction details), as implemented by the *Manual Steering Mode*. The user doesn't have to learn voice or gesture commands and to get used to the screen-based menu. Additionally, due to the close of coupling between robot and user this kind of interaction has a high degree of reliability because its independent from noise and light conditions. The user is able to apply experiences from a common, manual trolley to the robot trolley and is additionally supported by several functionalities like motor power, obstacle avoidance or local maneuvers triggered by *Force Commands*. For details on the construction of the final version of the *force sensitive handle* please refer to Appendix B.6 "Force sensitive handlebar".

**Interpretation of measured forces on the *force sensitive handle*** To achieve a high level of usability the trolley's handle must be able to generate movement commands out of the applied forces. The forces that the user exerts on the bars of the *force sensitive handle* are measured by eight strain gauges. The values are digitalized and transferred to an integrated DSP signal processing board. Here the eight measured forces are transformed into two forces and one torque. Additional signals are provided by micro switches on the handle that are used as dead man's switches. When released, these trigger the classification of *Force Commands*. Each identified command motivates some of the behaviors of the *advanced behavior repertoire* (Tab. 6.3). While at least one of the buttons (or micro switches in the new version) is pressed the current command is classified as manual steering and the processed forces are directly transformed into a velocity set-point

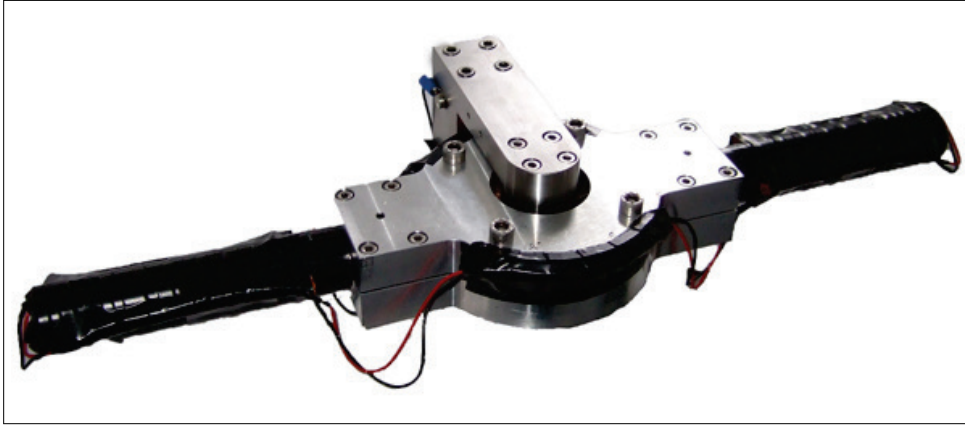


Fig. 6.5.: The *force sensitive handle* measures the forces applied by the user while the micro switches around it are activated. In the picture the third version of the handle is shown (see App. B.6).

Tab. 6.3.: Classes of *Force Commands*, their characteristics and the advanced behaviors triggered when classified accordingly.

Class name	R	FX	FY	T	AB
STEER	No	-	-	-	Steer
PULL	Yes	H(-)	S	S	Stay here
WAIT	Yes	S	S	S	Stay in area
ROTATE L/R	Yes	S	S	H(+/-)	Support loading
PUSH L/R	Yes	S	H(+/-)	S	Park
PUSH FRONT	Yes	H(+)	S	Sl	Drive forward

R: Signal emitted when the handle is released

F: Value of force in [X,Y]-direction. Values: High(sign) or Small

T: Value of torque. Values: High(sign) or Small

AB: Corresponding advanced behavior which is triggered

vector to be executed by the robot's *Behavior Network*. This procedure is described in Appendix B.6 "Force sensitive handlebar" and in more detail in [61].

**Force Commands:** *Force Commands* are assistance functionalities to ease local maneuvering with the robot. Figure 6.6 illustrates this at the example of loading a heavy crate into the basket: The robot is steered to crate, then the command to turn around is given, so that the lowest side of the basket faces the crate. After loading the crate the command to turn back is given.

The following list summarizes the developed commands and the resulting motion behavior.

**Steer:** The robot is basically controlled by the handle directly. This is the basic command that is active as long as at least one of the dead man's switches is pressed.

**Stay here:** Forces the robot to stay exactly on the spot. This command is indicated by a short and sharp backward pull of the handle before releasing it.

**Stay in area:** Tells the robot to wait at the current position. But the robot is temporarily allowed to leave the position for example to avoid a collision with an approaching moving obstacle. This is the standard command when releasing the handle.

**Turn to support loading:** Commands the robot to turn its lower front side of the basket towards the user to ease loading of heavy goods. This is a two-part command: when indicated by a short sharp torque on the handle before releasing it, the robot turns. When afterwards a dead man's switch is shortly pressed and released at once the robot turns back to its initial orientation.

**Park on side:** When indicated by a short sharp sideways force on the handle, the robot searches the corresponding side for obstacles and calculates a parking position.

**Drive forward:** The robot moves few meters forward to free the way. The command is indicated by a short sharp push of the handle just as if thrusting an ordinary shopping cart away.

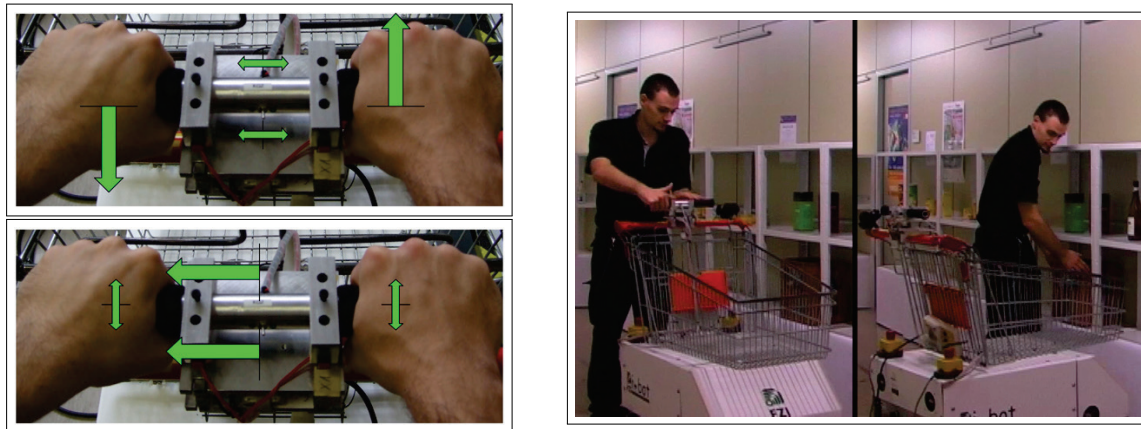


Fig. 6.6.: Left: Forces which trigger the “turn to support loading” command indicated by the green arrows (top) compared to forces for the “park on side” command (bottom). Right: The “turn to support loading” command in action.

**Obstacle Avoidance Assistant:** The velocity set point vector derived from the *force sensitive handle* is merged with the *reactive behaviors* for obstacle avoidance. This way, the robot is able to move around obstacles when the user does not notice them because he is either distracted or visually impaired. As always, the final velocity set point vector has to pass the safety behaviors so that collision-free movements are granted. A photo from such a scene can be found in Fig. 6.7.

**Shopping List Assistant:** Another assistance functionality can be implemented using the *communication layer* and the adaptive reactive behaviors, as was exemplary implemented using the *CR-UI*. When coming into the vicinity of a product on the shopping list the *communication layer* informs the control system about the product's position and the desire to slow down temporarily. When actually passing the product the control system slows down the robot and informs the *communication layer* that the robot is currently passing the product. The *communication layer* then utters a speech output “We are passing <product> from your shopping list”. A sketch of this behavior can be seen in Fig. 6.7.

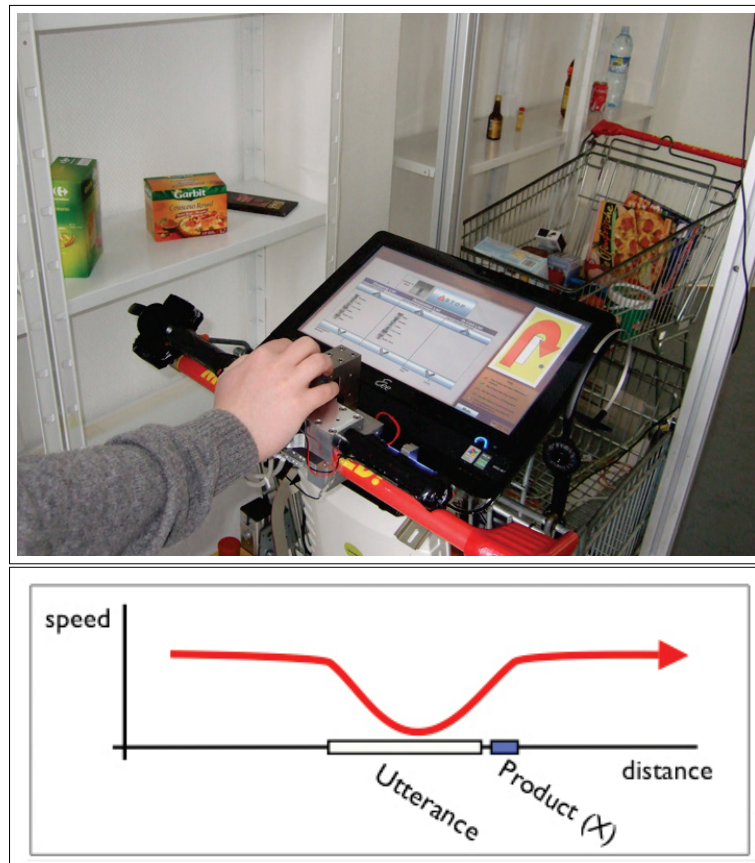


Fig. 6.7.: Assistants in *Manual Steering Mode*. Top: Photo showing a scene where *InBOT* is pushed by a distracted user with one hand only, so a collision with a parked shopping cart is imminent. Additionally, a design sketch of the path assistant can be seen illustrated by the red u-turn arrow on the touch screen. Bottom: Sketch of the functionality of the shopping list assistant (image source: [78]).

**Path Assistant** To inform the user about the path the robot is going to take, or to offer guidance during *Manual Steering Mode*, the robot's control system sends the next two representative goal points from the current task from the *task planner* up to the *communication layer*. The *communication layer* then visualizes this information with arrows on the touch screen. A design sketch illustrating the result can be seen in Fig. 6.7.

#### 6.4.2. Interaction based on the observation of the user

The second group of methods to control the *service robot* is based on the observation of the user. This way the robot gains knowledge on the user's relative position and his motion and can behave accordingly. This data is used to implement the *Servoing Mode*, the *Following Mode*, and the *Guiding Mode*. For this thesis two systems were used to observe the user and to acquire the needed data: the *intelligent environment* (see Appendix C.3 "The *intelligent environment*") developed in the course of this thesis as well as the *user tracking with onboard sensors* (see Appendix C.2 "Detecting and tracking the user using onboard sensors" and [63], [59]) developed at LAAS CNRS which was integrated to provide a wider basis for evaluation. The user's data is distributed in the control system via the *local world model* – either by adding the data directly



or by transferring it by WiFi to the robot first. This way, each behavior is able to choose the user's data as data input. Among others, this choice is triggered by the selected *mode of operation*.

**Visual servoing regarding the user:** For closely coupled local maneuvers of robot and user the *visual servoing* or the *Servoing Mode* is used. Here the robot does not only try to move continuously to the user's position while adapting to distance and pace like in the *Following Mode*, but is controlled in a closed loop by the user tracking system. The main purpose is to steer the robot in such a way that it always stays in the close vicinity of the user without impairing or blocking him. In the supermarket scenario this behavior is useful when the user is moving along local shelves while searching for a specific product. The robot here travels alongside the user and gives room when the user approaches the robot.

For the case where the user data is acquired by the integrated on-board camera system mounted on a pan tilt unit (PTU), control laws were integrated as well, aiming to control the PTU and the robot in unison as described in work by T. Germa at LAAS [59].

To achieve this behavior, the user's velocity is directly fed into the top fusion behavior of the *reactive layer* as velocity set-point vector. This way the robot imitates the user's motion while avoiding collisions using the *avoid obstacle* and the *safety behaviors*. The behavior for orientating the robot in a fixed direction is motivated and given to position of the user. Thus, the robot always turns the side with the touch screen towards the user.

**Following the user:** The *Following Mode* is designed for users who want to be in charge themselves. In the supermarket scenario these know where to find their product and want the robot just to follow. Accordingly, the user precedes the robot and the robot has to keep a defined distance using a position-based control. By activating the *Following Mode* the *follow object* behavior in the *tactical layer* is motivated and chosen in the fusion behavior. Additionally, the behavior is instructed to choose the user's position in the *local world model* as reference position which is to be followed in a defined distance – 1.5m has proven to be preferred by most users. The behavior reduces the distance only, it does not try to increase the distance if it should become too short: when the user moves towards the robot, the robot just stops, thus, enabling the user to put products into the basket. Again, the behavior for orientating the robot in a fixed direction is motivated and given to position of the user. As can be seen in Figure 6.8 the robot always turns the side with the touch screen towards the user, thus driving backwards.

**Guiding the user:** Here the robot is in charge and leads the user in *Guiding Mode* while adapting to his pace (Fig. 6.9). This is thought to be the standard mode of in the shopping scenario. The robot plans and moves along the route just like when operating in the *Autonomous Mode*. But the adaptive behaviors of the *reactive layer* reduce the current velocity of the robot if the distance to the user becomes too large – 1.0m has proven to be a suitable threshold here. If the distance becomes larger than 2.5m the maximum allowed velocity is set to zero – the robot stops and waits for the user. The behavior for orientating the robot in driving direction is motivated. The robot is driving forwards and thus turning the touchscreen towards the user.

**Lost the user:** When the user is lost, therefore the user cannot be detected anymore, the robot stops and waits for a defined amount of time. After the time has passed, the robot switches to *Autonomous Mode* and

Fig. 6.8.: *InBOT* follows a userFig. 6.9.: *InBOT* guides a user

plans a route to the last position where the user has been seen. This behavior is used only when operating in *Guiding Mode* or *Servoing Mode*. When in *Idle Mode* the user most probably wants the robot to wait and during *Following Mode* the robot will drive to the user's last known position directly without waiting.

#### 6.4.3. Interaction based on high level user interface

The two preceding ways of controlling the robot represent the indirect and often subconscious methods. These are important for sharing control and interacting with the robot. The methods of interaction centered around the high level user interface are actually methods of commanding the robot using mainly a touch screen as well as speech recognition and synthesizing.

On the robot *InBOT*, two user interfaces were used to implement the *communication layer*: the *InBOT-UI* (see Appendix C.4) developed in the course of this thesis as well as the *CR-UI* (see [87] and Appendix C.5) integrated to provide a wider basis for the evaluation of the overall system. At first glance the user interface provided by both is very similar. Both provide a touch screen interface which can be used to manage a shopping list and to give commands to the robot such as “Guide me to ...”, “Follow me”, “Meet me at ...”, and “Stop!”. Both provide a map with the highlighted position of the robot and the current target. In addition to the touch screen both provide speech output via speakers or a headset and both accept a bar code scanner as input. Figure 6.10 shows a scene where a user scans a product to add it to the list of shopped items (*InBOT-UI*) and Figure 6.11 shows a user trying to hit the “Stop!” button on the touch screen while walking – which has proven to be more difficult than had been imagined, even when using a large stop button.

But there are also some differences in the user interfaces. The *CR-UI* offers also speech command input and is able to detect some gestures when used together with the *user tracking with onboard sensors* system of LAAS. The *InBOT-UI* in contrast focuses more on the shopping experience and provides a personalization with a login for frequent users, a management of old shopping lists, and a database of recipes which can



Fig. 6.10.: User scanning a product using the bar code scanner attached to the touch screen user interface



Fig. 6.11.: User using the touch screen while following the robot

automatically add products to the shopping list or which can find a recipe matching to the current list and suggest additional items. Also, the work flow when following a shopping list is slightly different: The *InBOT-UI* adds a product automatically to the list of shopped items after reaching the product (when the user does not press “Cancel” in an corresponding dialogue), the bar code scanner is used to get information on scanned products or to add additional products to the list of shopped items. The *CR-UI* adds products to the list of shopped items when they have been scanned, only.

The various commands which can be given by the user are finally processed by an *application logic* and then transformed into commands which can be sent to the *strategic layer* via the *command interface*. Events received from the *strategic layer* are used to display information, to update the robot’s location on the map, or to utter speech output.

## 6.5. Tactical behaviors for adapting to and communicating with the user

This section presents a set of explicit behaviors dedicated to HRI – in contrast to large parts of this chapter where concepts and effects are described which emerge from a multitude of behaviors, the processing of various data sources, and the overall architecture. The behaviors are analyzing the local situation and if they generate actions, these are location-based. Thus, these behaviors extend the group of *tactical behaviors* introduced in Chapter 4.5 “The *tactical behaviors*”. In addition to supporting the robot’s navigation the *scene analysis* is also used to inform the robot’s user about certain circumstances relevant for him as well as to generate social behaviors. Keeping the extent of this work in mind, the following behaviors are not explained in all detail. There will only be a brief introduction on the functionality followed by an illustrating sketch.

Three main classes of behaviors can be identified in the context of user interaction (UI):

- Informative UI-centered behaviors
- Adaptive UI-centered behaviors
- Hybrid UI-centered behaviors

The informative behaviors do not perform any motion action themselves but inform the user about certain events (e.g. the robot’s path is blocked) or about the intentions of other behaviors which will result in

movements (e.g. re-planning due to a blocked topological link). The second class of behaviors generates movement actions to adapt the robot's behavior to the geometrical situation for user's benefit, such as finding a parking position at a product in a way that the robot is not blocking the product. Here no speech output is uttered because these behaviors shall work subliminal without bothering the user. The hybrid behaviors inform the user about certain events and explain the actions which are going to be performed. An example could be a situation when the robot cannot detect the user anymore while moving together with him. Here the robot will announce that it has lost the user and that it will start looking for him. Another quite frequent example is a situation where the robot has to start driving towards and finally to drive past the user. In this case the robot will utter a warning and wait a second for the user to react before starting to move.

#### TB: Look for Corners

<b>Situation:</b>	The next goal point is not reachable by the robot in a direct path
<b>Input:</b>	Goal point
<b>Data used:</b>	Robot position, robot shape, obstacles ( <i>object database</i> )
<b>Objective:</b>	Generation of sub-goal points that lead the robot around obstacles and enable the robot to avoid U-shaped obstacles / dead-ends.
<b>Output:</b>	First sub-goal point that leads the robot around the obstacle, trigger for speech output if the robot cannot find a path

This behavior was introduced when describing the navigation system in Chapter 4.5 "The *tactical behaviors*". It triggers a speech output if the geometrical analysis of the area does not produce a suitable path.

#### TB: Inform about topological re-planning

<b>Situation:</b>	It was detected that a topological area or link is blocked
<b>Input:</b>	Information that the next area or link to pass is blocked from the corresponding behavior (Chap. 4.5.5 "TB: Detection of Blocked Topological Links and Areas" and 4.5.5 "TB: Generation of Virtual Topological Areas")
<b>Data used:</b>	
<b>Objective:</b>	Informing the user about the re-planning of the route. If the user does not give any command in the meantime, after five seconds the re-planning begins
<b>Output:</b>	Event for the <i>communication layer</i> to trigger speech out: "Now way could be found. Do you want to take over or shall I look for another one?"

**TB: Inform that the user is blocking the robot**

<b>Situation:</b>	The path is completely blocked and the user is part of the blockage
<b>Input:</b>	Information that the area or link is blocked by the corresponding behaviors (Chap. 4.5.5 “TB: Detection of Blocked Topological Links and Areas” and 4.5.5 “TB: Generation of Virtual Topological Areas”), user’s position, target location
<b>Data used:</b>	<i>Object database</i>
<b>Objective:</b>	Stopping the robot and asking the user to free the path while showing the user in which direction the robot intends to drive (by turning in the intended driving direction). See Figure 6.12 for an illustration of the functionality.
<b>Output:</b>	Event for the <i>communication layer</i> to trigger speech out: “You are blocking me! Please stand aside.” Temporary sub-goal with the current position of the robot and the orientation towards the target.

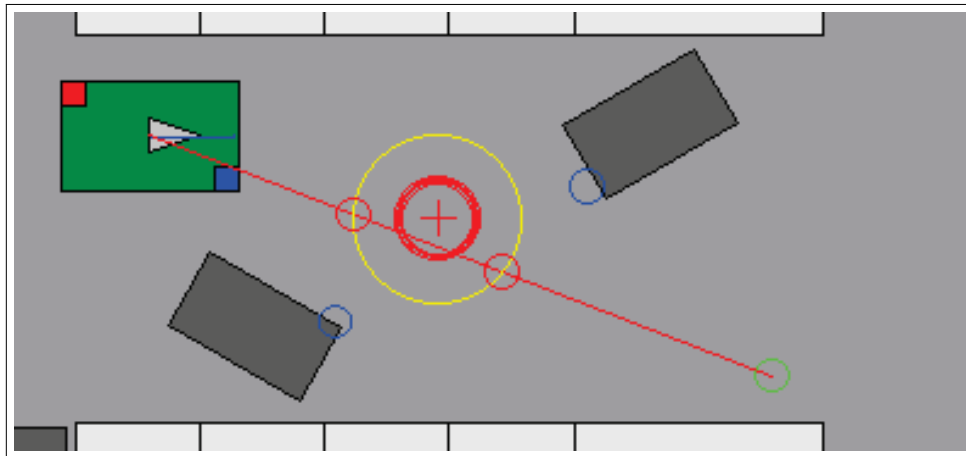


Fig. 6.12.: The user (red circle with cross) is part of a blockage of the intended path of the robot (red line towards green circle): the robot is not able to pass around the user without violating the social distance (yellow circle). This is identified by the fact that the distance between the obstacles’ corners (blue circles) to the yellow circle is smaller than the robot’s size.

**TB: Inform that the that the robot will drive past the user**

If the described situation is detected, the robot informs the user: “Watch out! I will drive around you” as sketched in Figure 6.14 and shown in the video snapshots in Figure 6.15.

A new sub-goal is generated where the X/Y coordinates are on the robot’s current position and the orientation points into the intended driving direction, which is acquired from the behavior for the geometric obstacle avoidance (TB: LFC). This way the robot shows the user in which direction it intends to drive around the user before starting to move. After few seconds this sub-goal is removed and the robot starts moving to its intended target using the geometric obstacle avoidance to move around the user.

6. User Interaction

<b>Situation:</b>	The user is located in the intended path of the robot, but there is still sufficient free space to move around (see Fig. 6.13 in comparison to Fig. 6.12)
<b>Input:</b>	User's position, target location
<b>Data used:</b>	<i>Object database</i>
<b>Objective:</b>	Asking the user to free the path and showing the user in which direction the robot intends to drive (around the user – by turning in the intended driving direction). Starting to move after waiting a short period of time.
<b>Output:</b>	Event for the <i>communication layer</i> to trigger speech out: “Watch out! I will drive around you”.  Temporary sub-goal with the current position of the robot and the orientation towards the next sub-goal point from the geometric obstacle handler (TB: LFC), therefore, the next point on a path around the user.

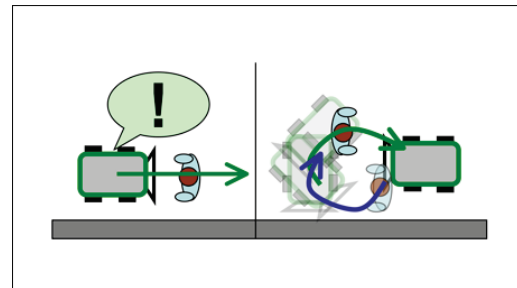
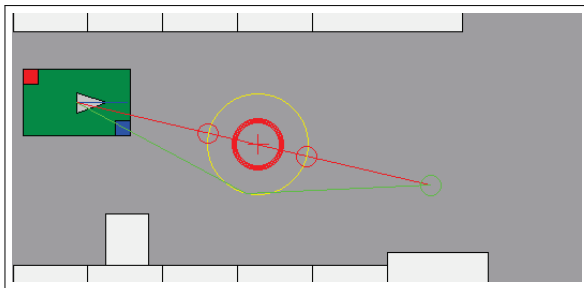


Fig. 6.13.: Scene where the robot detects that the user is in the intended path. The robot plans a path around the robot ensuring a social distance (yellow circle).

Fig. 6.14.: Sketch of the concept: the robot informs the user, waits for the user to make space, and then start moving.



Fig. 6.15.: Video snapshots showing the social behavior: After being commanded to guide the user to a product, the robot intends to move backwards. So the robot informs the user by uttering a warning and pointing into the intended direction. While the robot is turning, the user moves to the side to let the robot pass. Afterwards, the robot starts moving to the designated target and the user joins in. During the experiments, the users were not told that the robot would behave like this but almost all participants acted in a similar manner.

**TB: Take action because user is lost while guiding**

<b>Situation:</b>	The user is lost in <i>Guiding Mode</i>
<b>Input:</b>	Last user position
<b>Data used:</b>	
<b>Objective:</b>	After few seconds move to the last position where the user has been seen
<b>Output:</b>	Event for the <i>communication layer</i> to trigger speech out: “I have lost my user, I will start searching for him” Goal points identical to the last position of the user.

**TB: Take action because user is lost while following**

<b>Situation:</b>	The user is lost in the <i>Following Mode</i> or <i>Servoing Mode</i>
<b>Input:</b>	Last user positions
<b>Data used:</b>	Occupancy grid, <i>object database</i>
<b>Objective:</b>	Move to the last position where the user has been detected. If the current location is (already) the last user position, identify occluded areas in which the user could have disappeared and move to check these areas
<b>Output:</b>	Event for the <i>communication layer</i> to trigger speech out: “I have lost my user, I will start searching for him” Goal points from which occluded areas can be seen

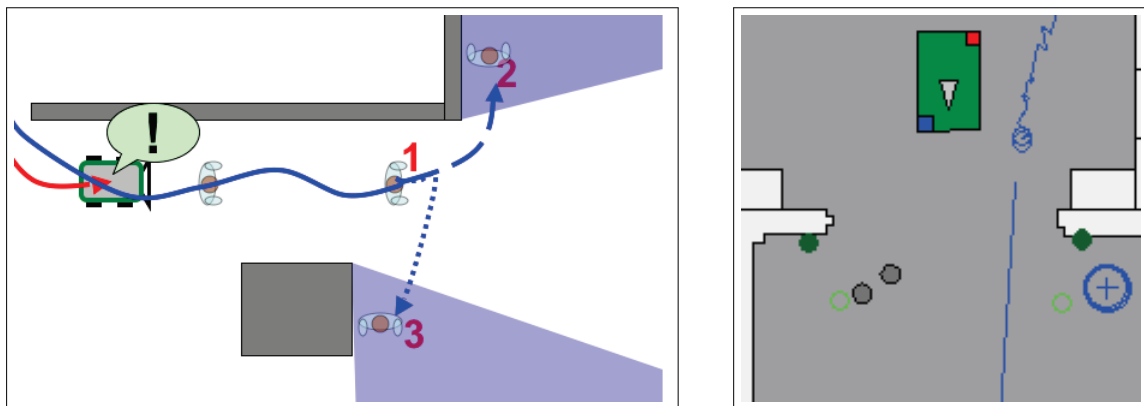


Fig. 6.16.: Left: Sketch of the behavior's functionality: the user has been detected the last time at position one and has therefore disappeared in one of the two areas the robot cannot see in (shaded in blue). The robot announces that it has lost the user and starts moving to the positions 2 and 3 to look for the user.

Right: Screenshot of the MCAGUI showing the path of the user and his last position (blue zigzagged path with small blue circles). The blue line indicates the estimated movement direction of the user and the large circle with the “+” in the center is the actual user position which cannot be seen by the robot (the control system has been given the on-board sensors' information only). The dark green blobs mark the closest corners of the occluded areas and the light green circles the resulting target points of the robot.

If the user is lost while operating in the *Following Mode* or *Servoing Mode*, the robot first moves to the position where the user was seen last. If the user is not found again, the robot searches the *object database* for obstacles in the close vicinity and identifies occluded areas in the occupancy grid (see Fig. 6.16 (left) for a sketch and Fig. 6.16 (right) for the behavior in action as documented in the MCAGUI). The behavior

generates goal points from which the occluded areas can be seen. It starts moving towards the occluded areas which fit to the estimated movement direction of the user best, while uttering a speech output.

### 6.6. Sharing and trading of control

After describing the individual components involved in HRI (e.g. modes, modalities, and behaviors), this section will focus on the question how the user can control the robot in general – in the most convenient way. The user does not want to bother keeping modes, modalities or path planning in mind. A very efficient way of interaction between user and robot emerges from the combination of the individual components and their organization in the control architecture: *Control sharing* – sharing instead of trading control.

A common way of controlling a robot is to give it a command and then to wait for the execution of the task. Sometimes the robot gives feedback or asks for clarification or acknowledgement and the user answers. This is performed in a clearly sequential manner.

But in complex situations this is not sufficient, especially when involving “first time” users or users who have to concentrate on a task on their own. A customer in a supermarket will not be happy having to explicitly control a robot all the time. One could think of examples like a user who is distracted by some surprising event, stops and forgets ordering the guiding robot to wait: robot drives away. Or parents shopping with a pram order the robot to follow. Someone moves in between them with his trolley, so the robot is not able to follow anymore. The robot sometimes needs means to take initiative and even to ignore or interrupt the last order to initiate necessary actions.

On the other hand, customers do not like the feeling of being driven by the robot (a statement heard by participants who preferred the *Following Mode* over the *Guiding Mode*). There has to be a mutual giving and taking of control to facilitate a convenient behavior which also prevails in complex situations.

A very important issue when talking about interacting with a *service robot* is the question of exerting control simultaneously or only sequentially – the question of sharing or only trading the control as it was named by T.B. Sheridan. When defining the requirements for the control system (Chap. 1.4.1 “R2 Human-Robot Interaction capability: user interaction”) it was defined that the control system shall facilitate sharing control between user and robot.

As discussed when motivating the control architecture, T.B. Sheridan described in the context of *supervisory control* the phenomenon of *control sharing* where an operator (supervisor) exerts control on some *control variables* of an automaton while the automaton itself has control over the remaining *variables* [147]. This contrasts the *control trading* where the user gives a command and the automaton executes the task autonomously having full control on all *control variables*. As also has been discussed, P. Griffiths and R.B. Gillespie discovered that putting the human and the automaton in a collective control loop improves the task performance because the human is able to focus on his given task without exterior disturbances [65]. In the shopping robot scenario the shopping process itself would be – according to the customers’ desire – the human’s task which should not be disturbed by trying to maneuver the robot through a cluttered corridor or finding the shortest way to a product. This control share is provided by the robot (based on the chosen *mode of operation*).



When combined, all the described capabilities, behaviors, and modalities together generate a situation where various environmental influences, including but not limited to the user (actively by tasks or commands and passively by his presence), exercise control on the robot. Accordingly, all these influences such as the robot's task planning, the user's behavior, and moving objects around the robot share control on the robot.

Modes of operation	Control			
	Safety	LN: Reactive Part	LN: Tactical Part	Global Navigation Task
Manual Steering Mode w/o OA	robot	robot	robot	user
Manual Steering Mode	robot	robot	robot	user
Servoing Mode	robot	robot	robot	user
Following Mode	robot	robot	robot	user
Guiding Mode	robot	robot	robot	user
Autonomous Mode	robot	robot	robot	user

Control by: robot user

Fig. 6.17.: Sketch showing the control shares regarding modes and navigation components: The columns indicate components of the navigation system (safety, local navigation, global navigation, and setting the task). They are shaded in red when in the current mode the robot owns the control on this component, in blue when the user is in charge. The less coupled the interaction in the *mode of operation* is the more responsibility is with the robot, the more coupled the interaction, the more the user is in charge.

Depending on the individual *modes of operation* the interaction between robot and user is more or less closely coupled. User and robot have to either share or trade the actual control. In the *control sharing*-case the coupling of user and robot in the control allows the user to perceive the robot's actions while allowing him to contribute his own intentions simultaneously. This is most significant while operating in the *Manual Steering Mode* or *Servoing Mode* but can also be recognized in the *Following Mode* and the *Guiding Mode*. In the *control trading* case user and robot take turns in giving and executing commands without interaction once a command is given. This can be seen during *Autonomous Mode* for example after a "Meet me at" command. Figure 6.17 shows which component of the navigation system is provided by the robot and the user, respectively, depending of the current *mode of operation*. The basic safety behaviors are always active and the *Avoid Obstacle* behaviors of the local navigation are almost always active (the only exception is a special mode using joysticks or the MCAGUI available for operators, only). The higher and the more abstract the navigation component, the more likely it is to be taken over by the human.

According to the classes of interaction depending on the available modalities, three major levels of *control sharing/trading* can be recognized: force-based *control sharing*, observation-based *control sharing*, and command-based *control trading*. Being able to include all modalities of user input is achieved by introducing the user's input into the top-down control data flow in an orthogonal manner: the corresponding data is fed into the data flow on top of each layer of the control architecture – commands into the *communication layer*,

position-based data into the *tactical layer*, and force or velocity-based data into the *reactive layer* (see Figure 6.3 and Chapter 3.2.3 “Incorporation of *control sharing*” for further explanations).

The next sections are describing the *control sharing* for the three classes: force-based *control sharing*, observation-based *control sharing*, and command-based *control trading*. These three sections are followed by a description of the impact of the *control sharing* on the robot’s motion during the individual *modes of operation*. The topic of *control sharing* is then concluded by the section describing experiments made with the robot.

### 6.6.1. Force-based *control sharing*

While operating in *Manual Steering Mode*, the coupling between user and robot is the closest compared to the other modes. The user exercises the highest degree of control on the robot. Usually the user has full control, but in dangerous situations the obstacle avoidance assistant can generate repelling velocity set-point vectors which are merged with the velocity set-point vector acquired from the *force sensitive handle*. If these two vectors point in opposite directions they cancel each other out. The robot will stop before hitting the obstacle. If they are orthogonal to each other they result in a combined movement: ahead in the direction the user desires and to the side to avoid the obstacle (see Fig. 6.18). This way the obstacle is avoided in a curved motion. The same mechanism counts for a gap finding assistant that eases the movement through narrow gaps and for the path finding assistant that indicates the shortest path to a product by slight direction changes. The user himself detects this self-control of the robot at once by feeling the feedback of the movement at the handlebar of the robot. He can either give or override the robot.

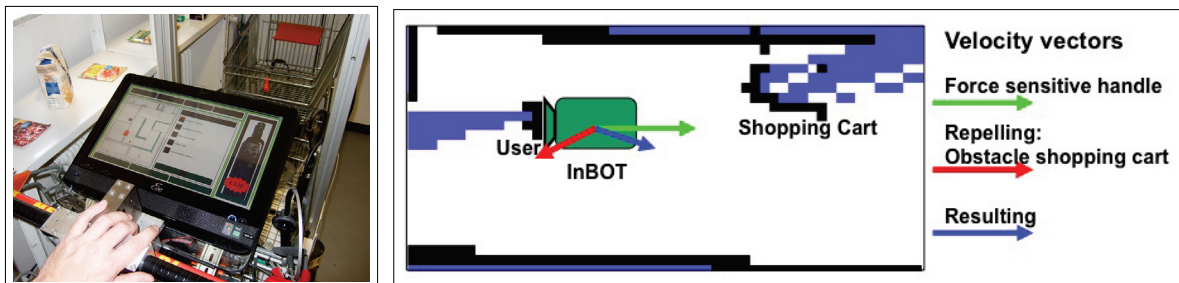


Fig. 6.18.: The user is steering *InBOT* just as an ordinary shopping cart – casually with one hand only – using the handle. Because steering with one hand only the user benefits from being supported by the reactive obstacle avoidance behaviors. Left: picture of the scene from the user’s point of view. Right: Occupancy grid showing the robot’s view on the scene. The influence on the robot’s actual motion is illustrated by the arrows. The green arrow is the velocity set-point vector derived from the forces applied on the handle, the red one is the repelling vector from the obstacle (AO v1), and finally the blue one is the resulting velocity set-point vector.

### 6.6.2. Observation-based *control sharing*

While operating in the *Servoing Mode* the coupling between user and robot is almost as close as in *Manual Steering Mode*. The user’s movements are observed and fed into a closed loop control which generates the motor commands. The user controls the direction and velocity of the robot indirectly. In both cases *InBOT* has the opportunity to contribute the obstacle avoidance to be merged with the movement demands of the user. Therefore, the robot can move around obstacles without forcing the user to control this action himself.

During the *Following Mode* the user is still dominant. He does not control the robot's movements in detail but he defines the general movement direction. The user walks normally and the robot follows the user in a defined distance. The robot continuously observes the position changes of the user to be able to control this distance autonomously. If the user is too far away it accelerates. But when the user comes closer, *InBOT* stops and stands still until the user has a defined distance again to enabling the user to load articles into the basket.

Even less coupled is the *Guiding Mode*. Here the robot performs a task such as moving to a given product self-dependently. It chooses the shortest path and generates motor commands according to goal directness and repelling influences from the obstacle avoidance. The user only controls the actual speed of the robot indirectly by his movements: Due to the demand that *InBOT* shall guide the user, it again observes the distance to the user. If the user comes too close the robot accelerates, if he gets too far away, the robot decelerates. Another possibility for the robot to take a larger share of control in this mode is to decelerate when trying to initiate a communication with the user, e.g. informing him about a product on the shopping list or special offers. These robot body movements to assist communication are presented in the collaborative work [78] together with TU Vienna and KTH Stockholm. In the context of initiating dialogues the field of "mixed-initiative" should be kept in mind as well, which is also of high relevance when talking about human robot interaction (see [21] for some own thoughts on mixed-initiative, human and robot responses, and time constraints as well as [69] for a collection of articles).

### **6.6.3. Command-based control trading**

The most decoupled interaction takes place in the *Autonomous Mode*. Here the robot performs a given task completely autonomous. The user is restricted to give a command and to rely on the robot's ability to perform the task to his satisfaction. His only method of interfering is to cancel the task or to interrupt it with a new one. The execution of a task is completely under the control of the robot. User and robot take turns in giving and executing commands and therefore trade the actual control.

### 6.6.4. Impact of *control sharing* while operating in the individual modes

This final section on the concept of *control sharing* describes the impact on the motion of the robot for each individual *mode of operation*. A sketch of the flow of control between robot and user is given in Fig. 6.19, data from experiments will be given next section.

In the top left part of the figure, the path of the robot around an obstacle is sketched, beginning at the left-hand side. The area shaded in red is the area where the robot will try to increase its control share, the area shaded in blue where the robot will release some of the control. In the remaining five parts of the figure, the individual flows of control are illustrated for the different modes of operation: the user's share in blue, the robot's share in red. In the first case the user simply steers the robot around the box, the control is totally with the user. In the second case the user steers the robot towards a target. When approaching the box the robot partially takes over the control to assist the user in moving around the box. Third the user orders the robot to follow him. The user moves to a target and the robot tries continuously to move to the user, so implicitly the user controls the movement of the robot. When approaching the box the robot ceases moving to the user only but takes the responsibility to move around the box autonomously. In the last two cases the robot is ordered to move to a target location which is performed self-dependent, with and without controlling the distance to the user.

This graphs illustrate how the *control sharing* is intended to work. In comparison, the subsequent section discussing the experiments presents a similar figure, showing the actual exerted control shares during the experiments.

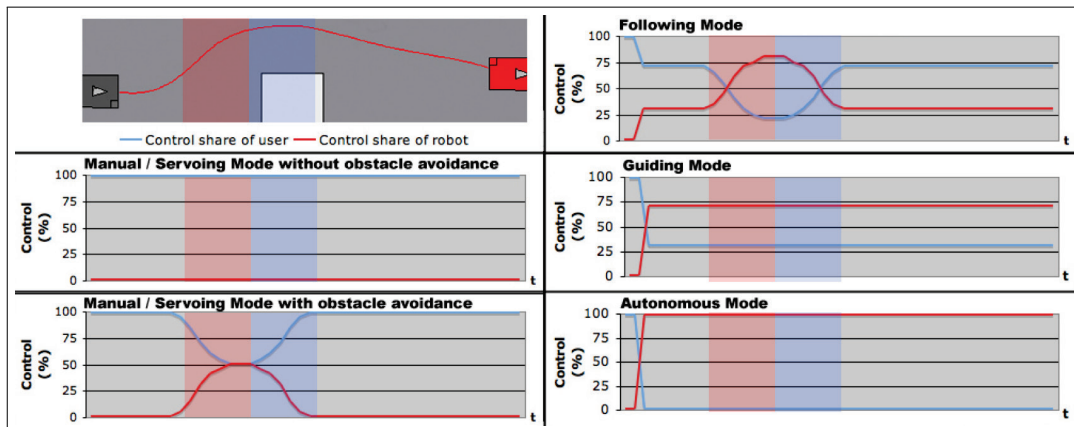


Fig. 6.19.: Concept of *control sharing* in the individual modes. Top left part: The robot moves from the left around an obstacle to a target position on the right. The control shares of robot and user are illustrated in the remaining five parts of the figure for the individual *modes of operation*: the user's share is marked in blue, the robot's share in red. The vertical axis is the amount of control the individual share contributes in percent, the horizontal axis represents the time or the distance driven, respectively. The area shaded in red highlights the situation when the robot will demand more control, the area shaded in blue the situation when the robot will release some control. The distance to the user in the *Guiding Mode* is assumed to be invariant.

## 6.7. Experiments

After introducing the individual components involved in HRI and the concept of *control sharing* which emerges from the components in conjunction with the control architecture, this section presents the outcome

of a series of experiments which was conducted involving users to put the HRI capabilities to trial. First, the performance of the implementation of the *control sharing* will be illustrated by means of two experiments, and afterwards general findings and the results from surveys will be presented.

#### **6.7.1. Experimental evaluation of *control sharing* and *control trading***

This section demonstrates the implemented concept of *control sharing* in action. Two of the experiments are referenced here in detail. The first one has been conducted with the robot *InBOT* and its user only, the second much more complex one involved additional persons with shopping carts. Both experiments will be accompanied by a sketch showing the individual control shares of robot and user changing over the time.

The control shares of robot and user are calculated as follows: The velocity set-point vectors of all motion behaviors are summed up in the two classes: user and robot. The task-oriented ones count for the robot (guiding) or for the user (following). The manual steering vector counts always for the user and the obstacle avoidance behaviors always count for the robot.

**Experiment with *InBOT* and user:** This paragraph presents an extract from an experiment involving the robot *InBOT* and its user. The following text describes the user's and the robot's actions. The text is accompanied by two figures: figure 6.20 shows the map of the mock-up shop with obstacles, products, and the paths taken by robot and user. The user was tracked using the *intelligent environment* (see Annex C.3). Figure 6.21 shows the control shares of user and robot during the experiment for each *mode of operation*.

The user is asked to pick six products (indicated by the Arabic numerals) in a given order and using a given *mode of operation*. The length of this shopping run adds up to 60m. The mode to be used is indicated by the Roman numerals: *I* for guiding, *II* for following, and *III* for manual steering. Special areas of interest – indicated by the letters – are highlighted as well. The user starts (A), ordering the robot to guide him to the products (1) and (3). On the way to (3) he recognizes the second product (2). The user diverges and goes to (2) while the (guiding) robot stops and waits at (B). The user takes the product and finally continues towards (3). When the user is again close enough to the robot, the robot proceeds guiding the user to (3). After taking product (3), the user orders the robot to follow him and heads for product (4) and (5). He crosses the hall with the robot following him. At (D) the robot cannot follow the user due to obstacles and waits for the user to return. When the user returns, they both move around a corner and a short time later around some obstacles (E). Eventually, the user stops at product (5). After taking the product, the user grabs the *force sensitive handle* and manually steers the robot back to to product (6), and then through a narrow door (F) towards the check out counter (G). The user – by order – almost hits the corner in front of the counter and then the counter itself, but the obstacle assistant clears the situation.

Looking at the control shares (Fig. 6.21) one notices that during the *Guiding Mode* part the robot is in charge until the user stops at (B). The control shifts to the user and the robots slows down. After the user proceeds, the control shifts back to the robot which proceeds as well. In the *Following Mode* part the control is with the user leading the robot like having a virtual leash. At (D) and (E) the robot is confronted with obstacles. Thus, it demands control to avoid them. In the last part (*Manual Steering Mode*), the control is in general with the user until he tries to pass a narrow door (F) or steers the robot into the checkout counter (G). Here again, the robot demands control to solve the situation.

Summarizing, it can be stated that the task was performed successfully and the *control sharing* enabled the robot to cope with the challenges of the task.

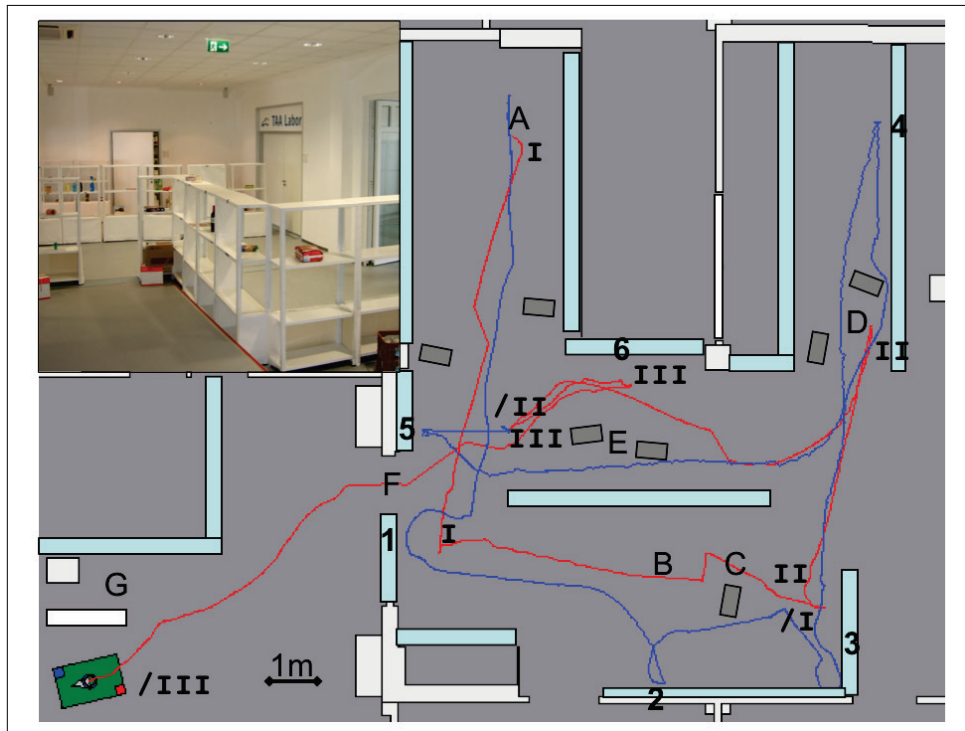


Fig. 6.20.: Map of the mock-up supermarket section with shelves marked in light blue, obstacles in grey, and the products to be taken with Arabic numbers. The paths of the robot (red) and the user (blue) start at (A) and end at the checkout counter (G). The modes used are indicated by *I*, *II*, *III* for guiding, following and manual steering, respectively. In the *Manual Steering Mode*, the user is always directly behind the robot, therefore no blue line is plotted here. Additionally, special areas of interest are marked with letters. Top left a small section of the shelves can be seen.

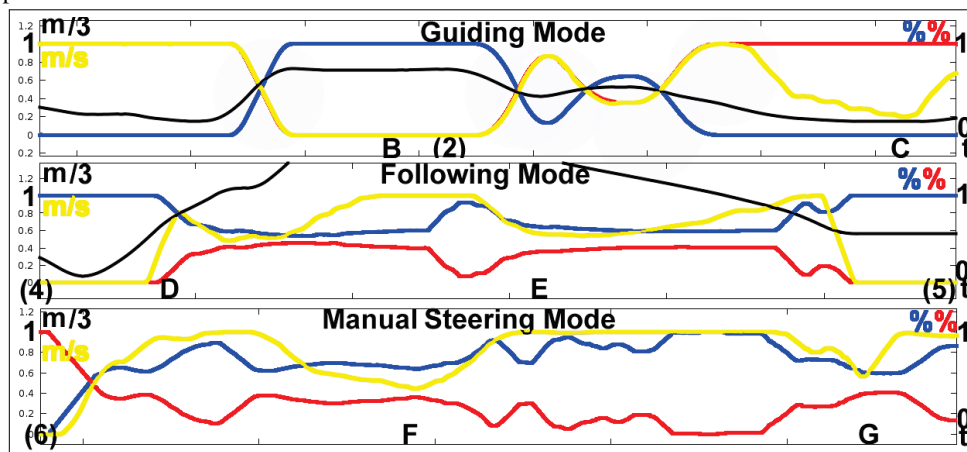


Fig. 6.21.: Diagram showing the control shares of user (dark blue) and robot (red). The velocity of the robot (yellow) and the distance between user and robot (black). In the first part (*Guiding Mode*) the control is with the robot as long as the distance between robot and user is small enough. When the distance rises (the user stops at (B)) the control shifts to the user, slowing down the robot (most of the time control and velocity have the same value, hence, the yellow and red lines overlay each other). In the second part (*Following Mode*) the control is with the user until the robot senses a threat by obstacles ((D), (E)). Here the robot takes control to avoid the imminent collisions, resulting in some variances of the robots velocity and distance to the user. In the final part (*Manual Steering Mode*) it is quite similar: the user has the majority of the control until approaching and finally passing a narrow door (F) or coming too close to an obstacle (G). Here the robot again takes over a larger control share to slow down and to change the direction of the motion.

**Experiment with *InBOT* user, and other shoppers:** An extract of the second experiment referenced here is shown in Figure 6.22 – again accompanied by a corresponding graph of the control shares in Figure 6.23.

This experiment has been much more populated, and so the figure displaying it is much more chaotic, too, showing the paths of several objects. Robot and user start at the bottom right corner. The user is instructed to get the products (1) to (5) in the following modes: 1: (G)uiding, 2: (F)ollowing, 3: (F), 4: (G), and 5: (F). During these tasks, three other – particularly reckless – shoppers (A), (B), and (C), operating ordinary shopping carts, cross the path of *InBOT*, forcing an reaction. The other shopping carts had actually been *ETrolley* and thus had an self-localization system on their own. This position information has been used during this test as the focus was on the usability and on the navigation system (especially for avoiding moving objects), not on the object tracking system. The user has been tracked using the *user tracking with onboard sensors* of LAAS (see. Annex C.2).

Again, the robot succeeded in all situations, even when impaired by its user and threatened by approaching shopping trolleys and imminent collisions. The control was shifted smoothly between robot and user so that the user did not have to be bothered with saving the robot from collisions. When the situations were solved, the robot – just as intended – automatically hands back the control to the user.

### 6.7.2. Evaluation of *InBOT*'s behavior by potential users

The section starts with general findings and evaluations of components from an HRI point of view. Afterwards, the results of surveys conducted with the users after the tests will be summarized.

**General and technological findings:** A general finding is that it is convenient for the user to have means of remotely commanding the robot – thus not having to walk to the touch screen all the time. Besides the obvious fact, that a remote method will reduce the effort of the user, it turned out that a touch screen interface is hard to use while the robot is driving – even if only driving slowly at a constant speed. Users tend to misjudge to position of the buttons and have a tendency to stop walking while pressing the button which increases the mentioned miss-judgment and makes them run after the robot in case of a failure. But the remote methods have their drawbacks as well:

- A speech recognition system is subject to robustness issues in such a noisy environment and with users which are not trained to pronounce the words “correctly” or even using local slang.
- Gesture recognition was at the time the tests have been conducted not sufficiently sophisticated to perform in a colorful environment with changing light conditions while robot and user are moving. From the sensors' perspective, recently there were very promising developments by the *Kinect* 3D sensor and the corresponding algorithms, which nowadays might have changed the picture significantly.
- Another method for remotely controlling the robot is using actually a (radio) remote control – for example a small GUI on a smartphone as was used in early tests (see Figure 6.24). The main drawback here is that the user needs a free hand to operate the smartphone. This cannot be done by a user who is carrying a bunch of products and wants to call the robot. Additionally, the usage of a smartphone GUI is more time consuming (unlocking, etc.) than just verbally telling the robot what to do. And finally, the WiFi-based localization of mobile devices can still be of bad quality.



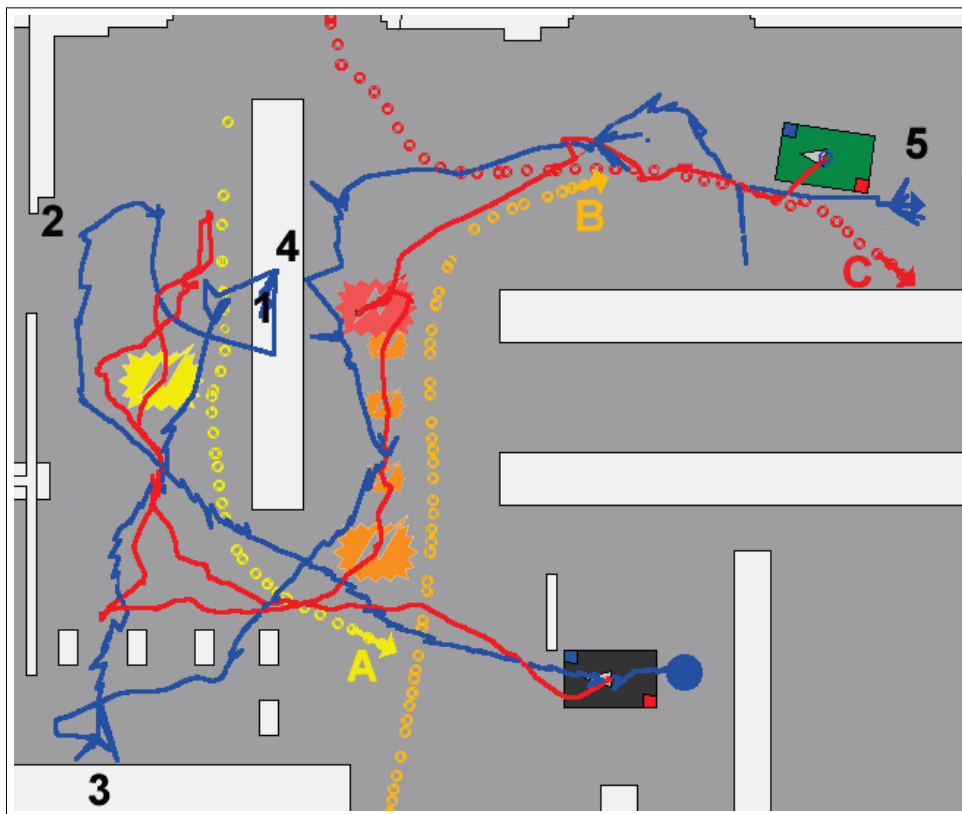


Fig. 6.22.: Shopping run with other shoppers: the robot's path is marked in red, the user's path in blue, and the other carts' path using the dotted lines with arrows showing the driving direction. Critical areas are indicated by flash symbols. *InBOT* and user start in the lower right corner (grey rectangle and blue dot). The user is guided to the first product (1) while avoiding the moving shopping cart (A) with a strong evasive motion to the left at the yellow flash symbol. After following to the next two products (2) and (3), the robot guides the user to product (4) and avoids the crossing cart B by slowing down and letting it pass. The robot continues by aligning itself behind the cart (orange flashes). While following the user to the final product (5), the robot's path is cut by shopping cart (C). Again, the robot waits and lets the cart pass (red flash).

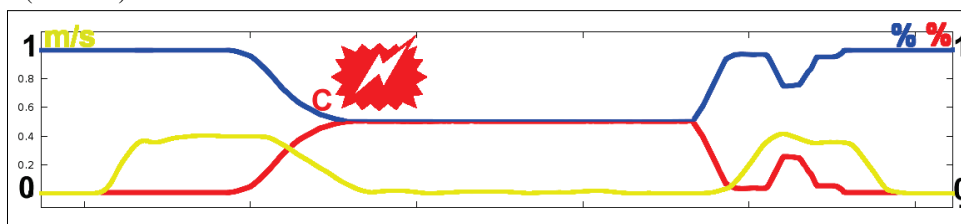


Fig. 6.23.: Control shares for a short sequence at the end of the shopping run: while following the user from (4) to (5) in *Following Mode*, the user exercises full control on the robot (control share of user: blue), but when approached by the cart (C), the robot takes over a share of control to avoid the collision (control share of robot: red) resulting in a direction change and temporarily stopping of the robot (velocity of robot: yellow). Afterwards, the control is handed back and the robot starts moving again.



Fig. 6.24.: Commanding *InBOT* using a smartphone GUI. The smartphone is localized using the WiFi network. The GUI is implemented by the MCAGUI and provides the basic functionalities such as “Stop”, “Come to me”, or “Continue”.

From a technical point of view, a general result of the tests is that neither the *intelligent environment* nor the *user tracking with onboard sensors* is an optimal concept for the acquisition of user information. The *intelligent environment* provides much more robust and accurate information but it is subject to a time lag when transferring the information via WiFi network, especially when the WiFi network is highly saturated. The *user tracking with onboard sensors* in contrast provides almost real-time data but the field of view is very limited, the user is occluded frequently and the distance estimation is less good for many reasons, such as a changing perspective of the robot, own motion of the robot, changing light conditions, especially when the robot is turning, and so forth. According to these findings a fusion of both methods would be desirable.

**Survey following the user experiments:** Two shopping experiments have been conducted with untrained participants recruited “from the street”. The experiments have been taken place at FZI in Karlsruhe in collaboration with TU Vienna and KTH Stockholm, utilizing the integrated components *CR-UI* and *user tracking with onboard sensors*. This section summarizes the outcome from a human factors’ point of view, technical results have been summarized in Chapter 4 “BBC: Navigation, Obstacle Avoidance and Safety”.

The first experiment involved 20 users each with 10 meters to travel and 6 products to get with a special focus to the shopping list assistant triggered by the *CR-UI*. The second experiment involved 12 participants acting as the robot’s users, for each user a accumulated distance of approx. 60 meters to travel, and the picking of 12 products (Fig.6.25). The participants have initially been given a brief introduction to the robot, an explanation of their task, and a shopping list on a sheet of paper. Interestingly, most of the participants never let go of the sheet of paper, even though they entered the shopping list into the management system on the touch screen in the beginning.

The participants have been asked to answer a questionnaire in the end. The questions had to be answered in a scale of 1(worst) to 5(best) points – the neutral rating is 3 points. The rating quoted in this section is the average points (p) given by the participants.

The 32 randomly picked users were mostly students (77%) with mixed background. The others were administrative employees. The gender mix was 70% male to 30% female. The most noticeably control-related results are summarized:

- Most participants found it easy to get used to the robot (4.6p) proving that the goal to implement an intuitive control system and user interface was successful.

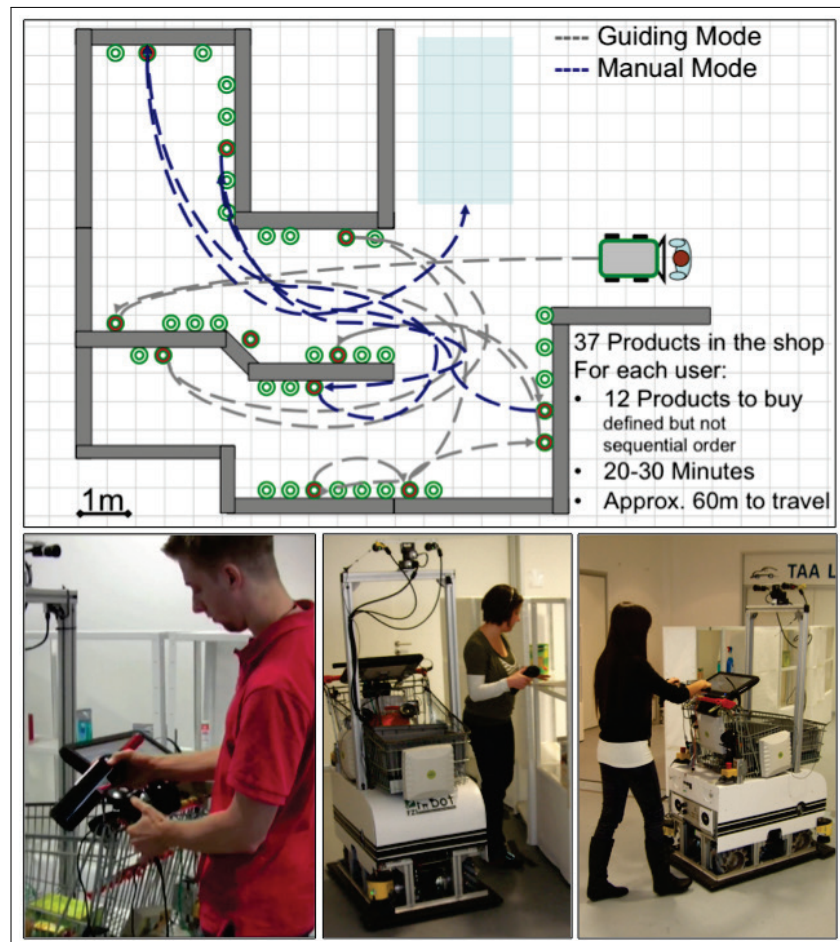


Fig. 6.25.: The user test: The top sketch shows the intended path of the robot and gives some general figures. The MCAGUI screenshot (bottom left) shows the path of the robot during for one exemplary user and the picture shows a user steering the robot *InBOT*.

- The overall rating for the robot's controllability and usability got 3.4 points each and the "feeling of being able to always control the robot" of 3.6p shows us that here is still room for improvements but no major fault.
- Only 3 users found it "hard" (2p) or "very hard" (1p) to actually control the robot.
- The guiding functionality was found to be very useful by the participants (4.6p).
- Finally, only 3 users said they did not feel supported well (1p or 2p) and all but two users found the robot to be a very interesting device.



## 7. Multi-Robot Coordination

This chapter completes the description of the developed control system. It shows that the designed behavior-based architecture is able to implement behaviors for multi-robot coordination.

In the context of the supermarket scenario there can be found two types of multi-robot behaviors. The implicit ones like solving deadlocks or traffic management are hardly recognized by the user. In contrast, the explicit ones have to be performed in cooperation with the user or are commanded by the the user. These are for example queuing up or following each other, building virtual trains. In this chapter some exemplary behaviors for the two groups are introduced. It is assumed that only few robots will be operating in a supermarket at the same time, hence, the focus will be on the explicit behaviors as the implicit ones are mainly needed for coordinating large numbers of robots. But even with few robot operating at the same time, these have to avoid colliding with each other and have to solve deadlocks at intersections. Therefore, these behaviors will be introduced as well, but not to the extent as it has been done for example by R. Regele [129] who developed a distributed algorithm for multi-robot path planning. The robots do not necessarily have to be of the same type – in fact very different robots have been used here. In particular *ETrolley* (see Appendix B.2) is not a real robot at all. It cannot take actions so the other robots have always to react to it, which has to be considered when resolving deadlocks and avoiding collisions.

**Scope of this chapter:** This chapter aims at demonstrating that the designed behavior-based control system is able to handle multi-robot situations, thus, the introduced behaviors are meant only as examples. They are only designed to the extent necessary to coordinate few robots in one place which do not have to actually cooperate. The behaviors are integrated into the *Behavior Network* by using the fusion behaviors and can be substituted easily. In contrast to the behaviors of the navigation system which are inspired by human motion patterns, the multi-robot coordination behaviors are directly focused on the application and thus not biologically inspired (i.e. flocks of bird or fish). One computer is chosen as moderator of the multi-robot system. It manages the data exchange and has some data processing capabilities, but the decisions are taken by the individual robots in an de-central manner.

**Organization of this chapter:** The first section describes the data handling between the robots which is a fundamental component for multi-robot coordination. The next three sections describe three types of behaviors: Sec. 7.2 the obviously necessary behaviors for collision avoidance and path negotiations among the robots, Sec. 7.3 an assistance behavior which enables the robots to build queues (e.g. at the check out counter or self service counters) freeing the user from the burden to stay with the robot all the time, and Sec. 7.4 a behavior for building virtual trains of robots – a set of robots follows a leading robot (almost) on the same path.

### 7.1. Sharing of the *local world model*

To enable multi-robot behaviors, the individual robots need to have access to the other robots' data. One robot (or any other PC in the same WiFi network) is selected as central moderator (the reader is kindly reminded of Figure 3.6 in Chapter 3.3.1). All robots register themselves with some data on the type of the robot when their control programs start up. In return, each robot receives an unique ID and a fixed basic priority. This priority is anti-proportional to the robot's degree of autonomy – i.e. giving a high priority value to robots with a low degree of autonomy. After connecting to the moderator, each robot periodically sends parts of its *local world model* to the moderator and downloads the same parts of the other robots' *local world models* and their corresponding IDs. Cycle times of 0.2 to 0.5 Seconds have given suitable results. The transmitted parts are:

- Current position, orientation and velocity, sparse list of past poses
- Current goal
- Queue the robot is listed in as well current position in queue
- List of moving objects and corresponding characteristics
- Difference between the current topological model to the initial topological model of the environment
- Current *mode of operation*

The structures of the *local world model* containing large amounts of data are not shared (occupancy grid map, laser scanner measurements, list of static obstacles, and so forth).

### 7.2. Multi-robot path arranging behaviors (RB, TB)

To avoid collisions between robots and to solve deadlock situations at intersections or bottlenecks, the robots need to arrange their paths among each other. Basically, the same mechanism is used as for the handling of moving objects described earlier: the robots transmit their position and velocity, thus, the robots can treat each other as moving objects with a given movement model.

To avoid deadlocks, the question which robot has to avoid which one is solved as follows: For each robot a individual and unique priority  $P_{Robot_i}$  is calculated:

$$P_{Robot_i} = M_{Robot_i} + B_{Robot_i} + \frac{ID_{Robot_i}}{ID_{Max}} \quad (7.1)$$

With  $M_{Robot_i}$  equal to the current *mode of operation*: (4: *Manual Steering Mode*, 3: *Following Mode* and *Servoing Mode*, 2: *Guiding Mode*, 1: *Autonomous Mode*) and  $B_{Robot_i}$  the basic priority, expressing the possibilities of mode changes. The robots' ID is used only to generate unique values in case of two robots operating in the same mode. Obviously, more sophisticated models involving the free room around the robots can be applied here for better results, instead of using only the level of autonomy.

Always the robot with the lower priority value has to avoid the one with the higher value. In the shopping scenario for example *ETrolley* as “non-robot” can only operate in the *Manual Steering Mode* and, thus, never has to avoid other robots – simply because it is not able to.

The implementation is straight-forward: the behaviors for avoiding collisions with moving objects described earlier take those robots into account, which have a higher priority value than the robot they are executed on.

### 7.3. TB: Queuing up

For the places in the shop where queuing up robots is desired like at the checkout counters, a queue data structure is shared between the robots and the moderator which contains the IDs of all queued robots. This structure is exchanged between the individual robots by WiFi. The structures are used to determine the robots' individual position in the queue. To request a position in the queue, a message "QUEUE" to the moderator is defined, which adds the robot to the specific queue. When the robot leaves the queue it sends a message "DEQUEUE" to the moderator, and then it is deleted from the queue. The queue position number of the succeeding robots is decreased, accordingly. Besides sending the messages, the *tactical behavior* generates a local goal point at the robot's designated waiting position.

In Figure 7.1 the green robot wants to queue up at the checkout counter. It sends the "QUEUE" message and receives its position in the queue. This position is used to calculate the goal point, where the robot will drive to. When the robot reaches this goal position it waits until one of the robots in front of it leaves the queue. When in this example the first robot leaves the queue, the following robots are assigned a new position and they calculate new goal points.

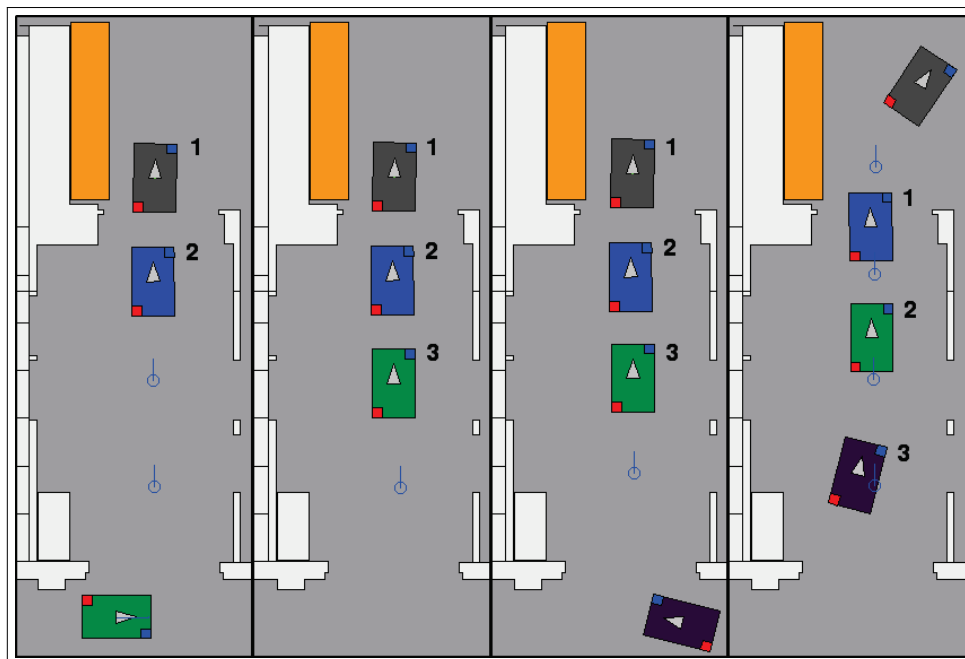


Fig. 7.1.: Queuing up at the check out counter (in simulation, only, due to a lack of available robots): the available queue positions are marked with blue circles (the line indicates the fixed orientation) and the robots which are already part of the queue are numbered. From left to right: 1: two robots are waiting, two positions are available, and one robot is approaching the queue; 2: the robot added itself to the queue; 3: a fourth robot approaches the queue; 4: the first robot left the queue freeing its position. The succeeding robot then leaves its position to move to the first one and so forth.

#### 7.4. TB: Virtual train

In addition to following the user, the ordinary following behavior can also be used to follow other robots. It gives the robot quite some freedom to decide how exactly to follow the target. But in multi-robot scenarios it might be of interest to follow a target more precisely. The tactical behavior *Virtual Train* lets the robot follow the target on the path it has taken (as long as the obstacle avoidance behaviors do not intervene). This might be useful only in the ordinary shopping scenario for maintenance tasks. But in related applications it can be indeed useful even for the customers: thinking for example of hardware stores which are not far from the supermarket scenario where people tend to buy larger amounts of goods. An additional benefit of this behavior is that in the future other devices can be coupled to the shopping robots: for example an automatic wheel chair would be able to lead or follow the robot.

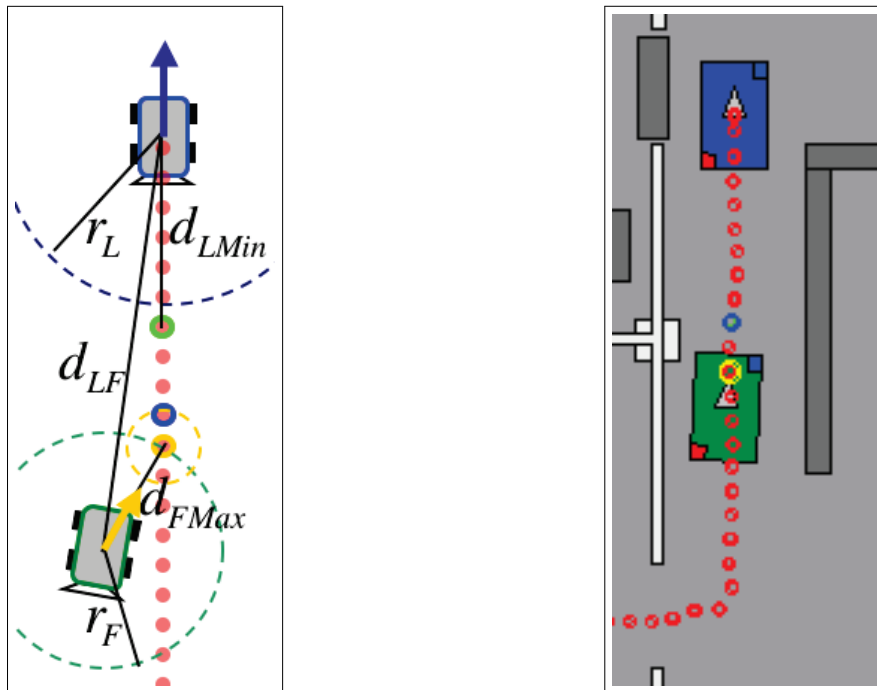


Fig. 7.2.: The virtual train concept.

Left: The Virtual Train concept: the (L)eading robot (blue) communicates a trail of past positions (red dots). The following robot (green) generates target points on this trail to follow the first one on the same path. The target has to have a distance of  $d_{LMin} > r_L$  from the leader and should be as far as possible in front of the (F)ollower, but not farther away than  $r_F$ .

Right: Screenshot showing a ordinary case of one robot following the other on the communicated trail (red dots).

In order to enable other robots to follow, each robot generates a sparse trail of past positions in a certain interval and communicates this trail to the other robots. The following robot follows this trail. As the behavior is a *tactical behavior*, it operates based on goal positions which are then executed by the *reactive behaviors*.

When a robot is ordered to build a virtual train with the other one, the trail of the (L)eaders is processed by the (F)ollower. Two special points on the trail are identified (see Fig. 7.2, left): First this is the furthest point on the trail which the F may moved to. It has to be further away from L than a defined safety distance  $r_L$ . The trail's points within a distance from L smaller than  $r_L$  are marked as invalid and can't be chosen



as goal points. So the following robot always keeps the distance to the leading robot. The second one is a point on the trail close to F, which is used as interim goal point for F on its way to the final goal point. It is as far towards L as possible, but has to be closer to F than a certain activation distance  $r_F$ . Additionally, a smoother orientation set-point can be generated by taking the average orientation from the next trail point and a point on the trail further ahead. Fig. 7.2 (right) shows the concept in application.

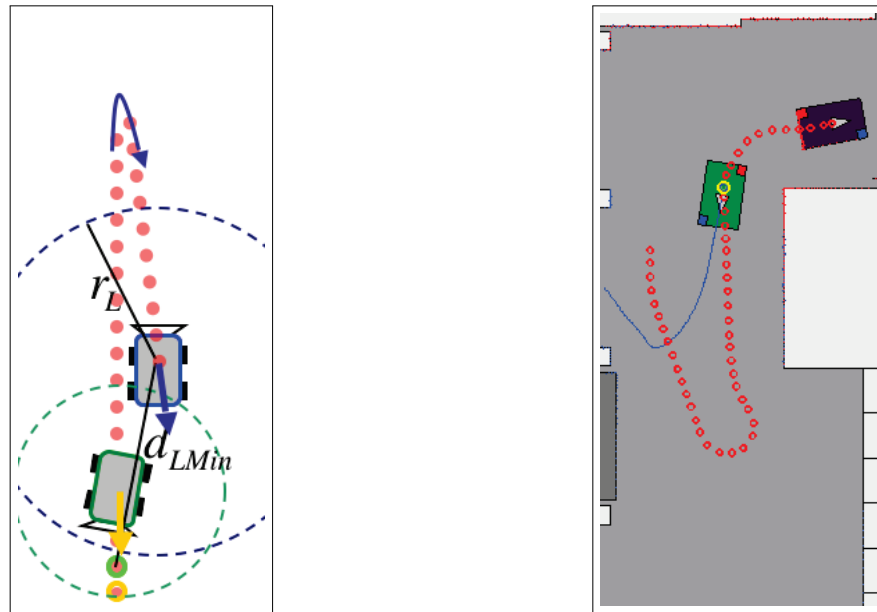


Fig. 7.3.: Left: Special situation regarding the virtual train concept: the leader turns around and needs space to do so. This way the goal for the follower is virtually pushed behind it, letting the follower retreat from the leader and freeing the path.

Right: The green robot is taking a shortcut to the other side of the trail once the circle  $r_F$  intersects with the trail. This way the robot omits moving a misleading indirection.

When L decides to move back on (almost) the same path, it can deactivate trail points which F has already reached. Due to the deactivated points near L, the candidates for F's next goal are pushed backwards on the trail and eventually they even move behind F. This way L is able to virtually push F back on the path it came. Figure 7.3 (left) shows a sketch and 7.4 an example for “the backwards driving” of a virtual train with *ETrolley* leading (the blue robot) and *InBOT* following (the green robot). Using this behavior, F cannot block L or even trap L in a dead end.

The points on the trail are always prioritized by their age, meaning that newer points are favored. Thus, the newest point inside the circle with the radius  $r_F$  is selected as the next goal point. This comes in handy when the target robot moves back close by the trail traveled earlier. The follower is able to take a shortcut as depicted Figure 7.3 (right). This is of special importance when earlier L was virtually pushing F backwards.

Finally, Figure 7.5 shows an example from a simulation with a virtual train consisting of 5 robots.

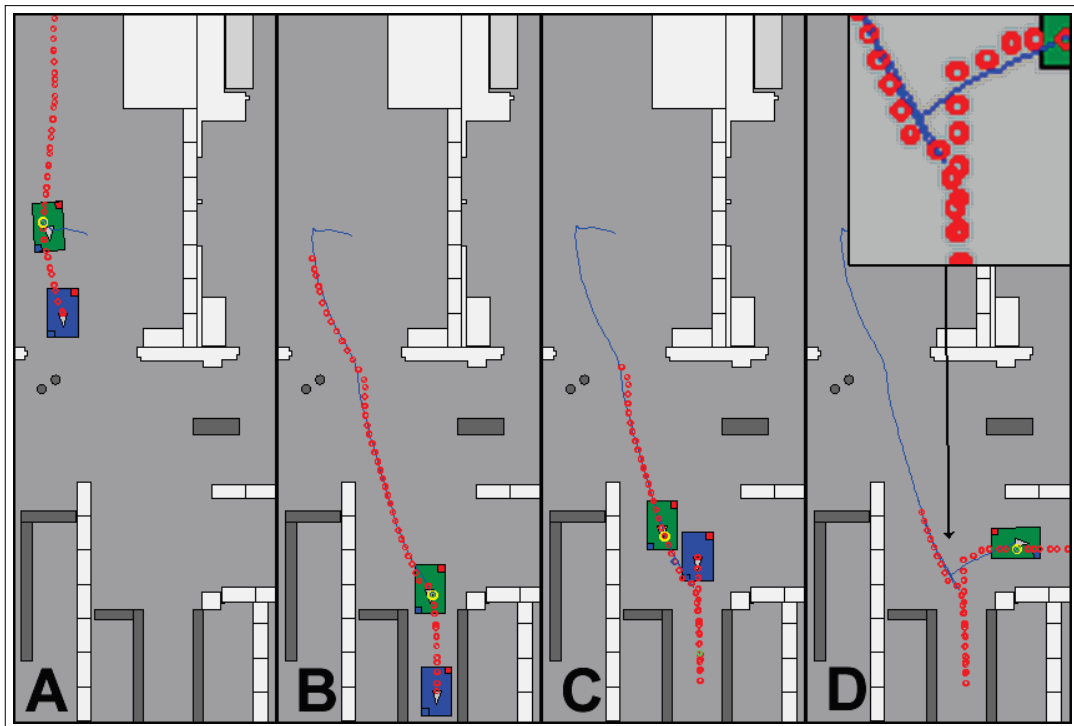


Fig. 7.4.: Forming a virtual train: In this sequence the blue robot is controlled manually (in fact it is *ETrolley*) and the green one is ordered to follow it shortly after it is passed by the blue one (A). After receiving the order the green robot moves on the path taken by the blue robot previously (red marks). In (B) the blue robot drives into a narrow passage, followed by the green robot. Later in (C) it drives backwards out of the passage virtually “pushing” the green robot back on the path it came. Finally (D) shows again the green robot following the path of the blue one. The blue line marks the path of the green robot. It shows how the robot was pushed back and that it later on follows the blue robot again without making the indirection of driving into the narrow corridor (even though the path of the blue one leads there).



Fig. 7.5.: Forming a virtual train: 5 robots are starting tightly packed with the order to build a virtual train. As they start moving they each wait for their preceding robot to gain a defined distance before starting themselves.

## 8. Conclusion, Discussion, and Open Issues

This chapter concludes this thesis. First of all, the work presented in the preceding chapters is briefly summarized. Then it is described how the control architecture has been implemented on several robot systems in addition to the implementation on *InBOT*, demonstrating the generalized character of the control. In Section 8.3 several aspects of the control system and its achievements are discussed. The section is followed by a summary of open scientific issues (Sec. 8.4) and finally by a list of general thoughts to be addressed when rolling out the system for the public (Sec. 8.5).

### 8.1. Summary

This thesis demonstrates that a service robot's control system based on the biologically motivated *Behavior Networks* performs well in complex scenarios involving a dynamic environment and human robot interaction, while providing significant benefits in orchestrating existing abilities and integrating new functionalities. The control system is aimed at scenarios where a robot is able to – and has to – navigate freely and where only basic environmental knowledge, i.e. no up-to-date global metric map, is available.

During the development of the *behavior-based* control system, three major issues were addressed:

- A hierarchical control architecture has been developed which combines the biologically inspired *Behavior Networks* in the lower layers with planners in the upper one. The main inspiration for the architecture as well as for the navigation system is derived from human motion patterns.
- A navigation system has been developed which is able to cope with highly dynamic environments. It consists of reactive behaviors for safety and collision avoidance, a geometrical scene analysis, behaviors dedicated to handling moving obstacles, and finally a *topological navigation* which facilitates navigating without relying on a global metric map.
- To tackle the chosen application – shopping assistance in a supermarket – the three core layers of the general control system are framed by the two application- and platform-specific layers, respectively. These manage the interaction with the robot's user on the one hand and control the robot platform *InBOT* on the other hand which was designed in the course of this thesis. Orthogonal to the usual top-down control data flow, the concept of *control sharing* has been incorporated into the architecture to facilitate cooperative task execution of robot and user.

As a first step, high-level requirements for the control system were defined, partially based on a survey as described in Chapter 1.3.3. These span the field of human-robot interaction as well as the core topics of navigation and localization. The high-level requirements define the main characteristics of the control system and the abilities of the robot e.g. the available *modes of operation*, the user interface, the necessity of *control sharing*, as well as the need to be independent from a global metric map.

The next step was to define the control architecture (Chap. 3). It is inspired by human motion behaviors as modeled by J.P. Hoogendoorn, the biologically inspired *Behavior Networks* by J. Albiez, and the concept of *control sharing* described by T.B. Sheridan. Based on these three sources of influence and the eight high-level requirements a hierarchical architecture was designed that provides a generalized control system in the three middle layers (*strategic*, *tactical*, and *reactive layer*). These are framed by the application-specific *communication layer* on top and the platform-specific *Hardware Abstraction Layer* below. Interfaces have been defined to facilitate substituting individual layers depending on the application's needs. Each of the three core layers uses a common "language" shared by all behavior modules contained. This shall ease hooking in new modules as well as incorporating input orthogonally to the usual top-down control data flow – a requirement defined for the implementation of *control sharing*.

The navigation system has been designed following the definition of the architecture (Chapters 4, 5, and 7). It consists of two parts: firstly the *global navigation* in the *strategic layer* based on a *topological navigation* and secondly the *local navigation* in the *tactical* and *reactive layer* based on the *Behavior Networks*. Again inspired by human motion patterns, the behaviors making up the more deliberative *tactical layer* perform a geometrical scene analysis and adapt the commands from the upper layer to the real environment. Additionally, the multi-robot coordination mostly takes place here and a spatio-temporal planner provides mid-term paths to avoid collisions with moving obstacles. The behaviors of the *reactive layer* are responsible for avoiding collisions with static and moving obstacles as well as adapting the robot's behavior to the user, thus, for instance enabling following and guiding behaviors. These behavior modules are only working on a limited amount of data to grant fast reaction times. This in particular counts for the safety behaviors at the very bottom of the hierarchy.

The human-robot interaction capabilities of the system result from a combination of some dedicated behaviors and the multimodal user interface, as well as from the design of the control architecture itself. The available levels of autonomy range from full manual control to full autonomy and are provided by the *modes of operation*: the *Autonomous*, *Guiding*, *Following*, *Servoing*, and finally the *Manual Steering Mode*. While operating in all but the *Autonomous Mode*, user and control system are able to exert control simultaneously and with equal priority, facilitating true *control sharing*. During the *Autonomous Mode* only the traditional *control trading* via the *communication layer's* user interface is available. In addition to the human-robot interaction, the system possesses basic features in multi-robot coordination like collision avoidance, building queues and *virtual trains*.

The control system has been evaluated in several tests. These were performed for each individual component, as well as for the complete system even involving users recruited "from the street".

### 8.2. Implementation on multiple robots

As discussed in the previous chapters, the control system and its individual behaviors have been tested and evaluated on the robot *InBOT*.

In parallel, parts of the control system are being, or have been, integrated in other robots to offer their specific functionality to the robot or their specific application. The robot systems and the individual contribution to their control will be summarized in the next sections, an introduction to a selection of the robots themselves can be found in Annex B. At this point I want to take the opportunity to thank each development team for being allowed to present their systems here.

### 8.2.1. Implementation on *InBOT*

The complete control system as it has been described in this thesis was implemented on the robot *InBOT*. The software implemented in the *MCA2* framework includes 18 groups and 94 modules including 32 behavior modules, the remaining modules are auxiliary modules aimed at coordination, data processing, and so forth. Additional *Hardware Abstraction Layers* have been implemented for *ETrolley* and the *CommRob Demonstrator (CoRoD)*. Two alternative versions of *communication layers* (including the multimodal user interfaces) have been integrated, one developed in the course of this thesis and the other one by TU Vienna [87]. Furthermore, two alternative systems for tracking the robot's user have been integrated, one developed in the course of this thesis using environmental cameras and optional laser range finders and the other one by LAAS CNRS using on-board cameras [59]. Figure 8.1 gives a brief overview of the main components.

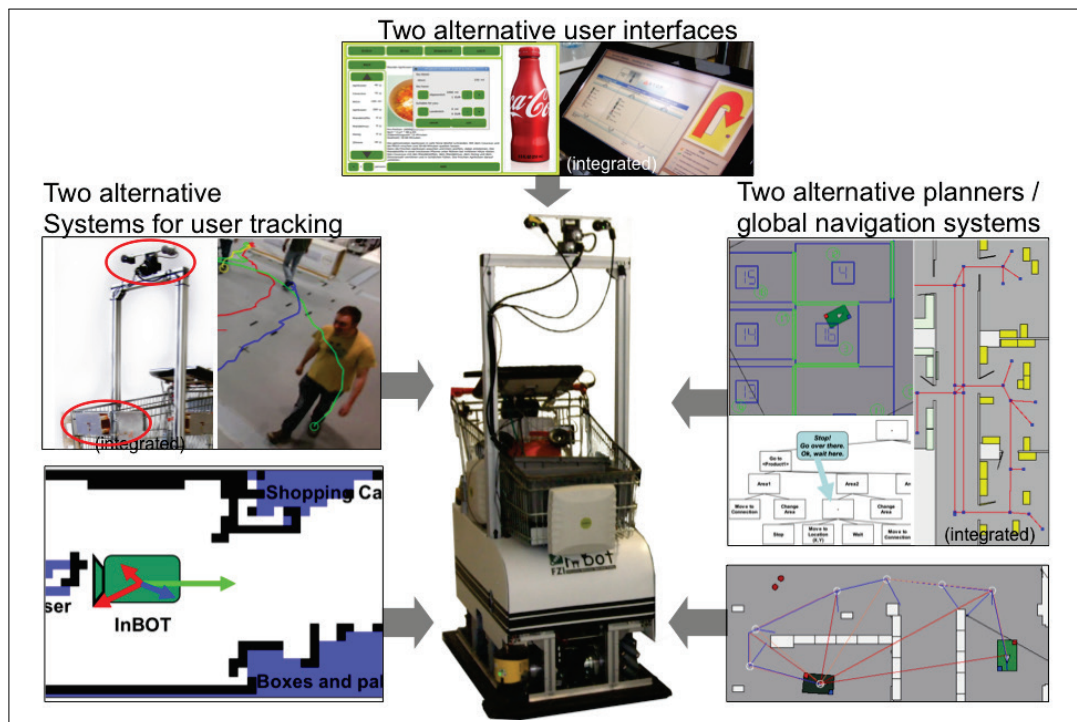


Fig. 8.1.: The control system as it has been discussed throughout this thesis has been implemented on the robot *InBOT*. In addition to the components developed, several alternative “third party” components have been integrated [59][87][144].

### 8.2.2. Implementation on *ETrolley*

The system *ETrolley* (Fig. 8.2) is not actually a robot as it lacks the motors to act. It is a shopping assistance device which is based on an ordinary shopping cart and implements the passive components of *InBOT* such as, but not limited to, the self-localization, user interface, shopping list management, and so forth. More information on the system can be found in Annex B.2. Obviously, the higher layers have a larger contribution here, the *reactive layer* has been omitted completely.

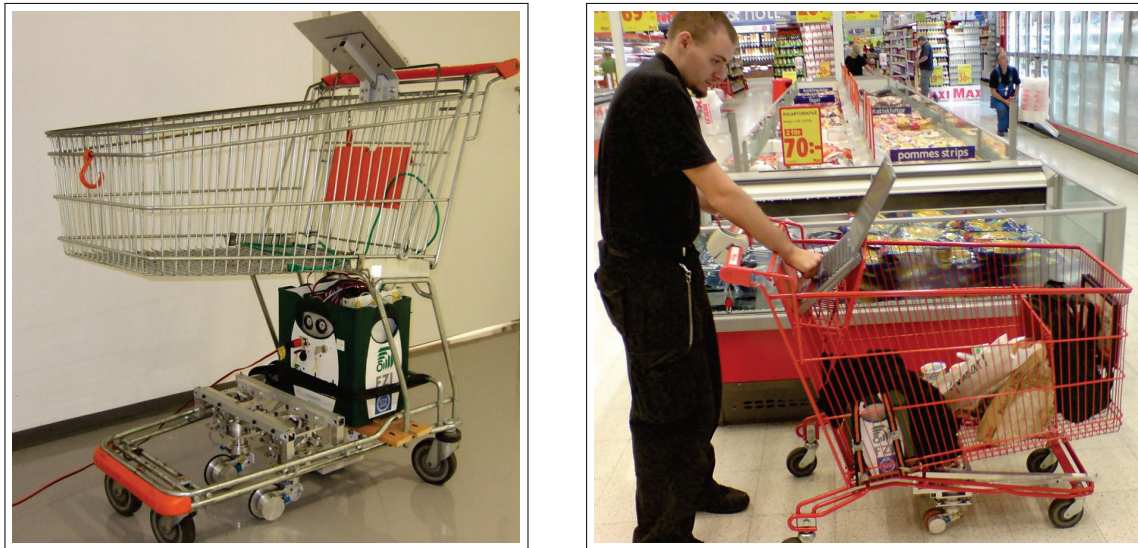


Fig. 8.2.: The system *ETrolley* incorporates all components of *InBOT* which do not depend on the drive system, for example the shopping list management and navigation assistance. It is equipped with a touch screen, a PC hidden in a crate, the measurement wheels for odometry, and the RFID reader. Two versions have been set up: one for tests in the FZI labs (left) and one reduced version for tests in a ICA Maxi supermarket in Stockholm (right, thanks to Helge Hüttenrauch for this opportunity).

### 8.2.3. Implementation on *Odete*

The robot *Odete* (Fig. 8.3) performs transportation tasks. It is based on a differential drive and uses a planar laser range finder in the front. Before the robot *InBOT* was finished, *Odete* was used as the development platform for the *Behavior-Based Control*. For example the first version of the *Avoid Obstacle* behavior was first developed on *Odete* and later ported to the holonomic *InBOT*.

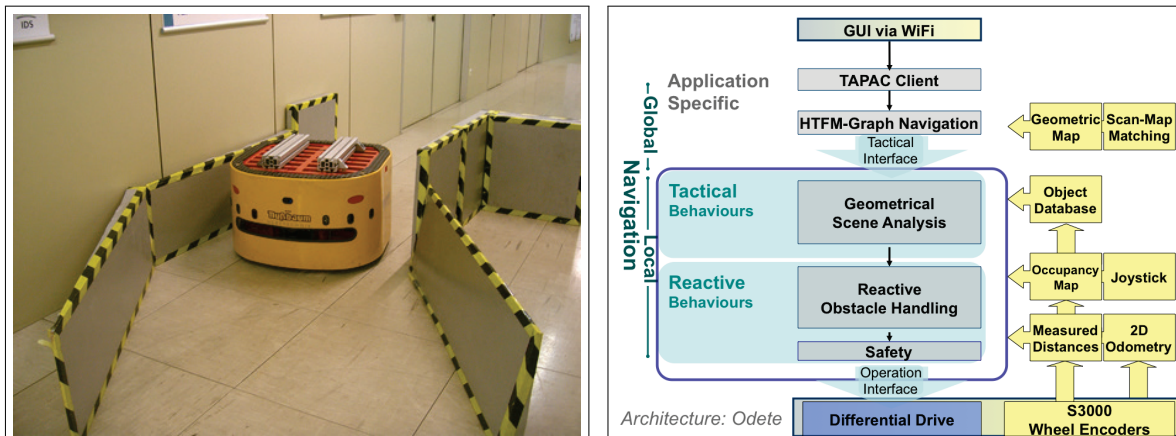


Fig. 8.3.: The robot *Odete*. Left: Passing a narrow and bent corridor. Right: the control architecture showing the tactical and reactive components of this thesis and the integrated application-specific components which have been present already at IDS [144] (see Annex C.6 and C.7 for a description).

As the main method of instructing the robot is network-based, and the transportation tasks are performed based on a graph, only the *tactical* and *reactive layers* have been applied – the already present programs for the (global) graph-based navigation and the logistics task management [144] were kept (see Annex C.6 and

C.7). The application-specific part is therefore extended to the global navigation.

*Odete* shows that the control system is able to operate with different programs contributing the global navigation and can work with different drive systems (but still a module is needed that transforms the 3D vector into a 2D one, consisting of the forward velocity and a circle's radius), and does not depend on a complete occupancy map – *Odete*'s FOV is to the front, only.

#### 8.2.4. Implementation on LAURON

The walking robot *LAURON*<sup>1</sup> (Fig. 8.4) – or more precisely the robots – have a long history at FZI, reaching back to 1992. The gait is inspired by the stick insect and implemented by the *Behavior Networks* of J. Albiez [3]. For the navigation and the task planning of the system several approaches were used (e.g. [56], [179]).

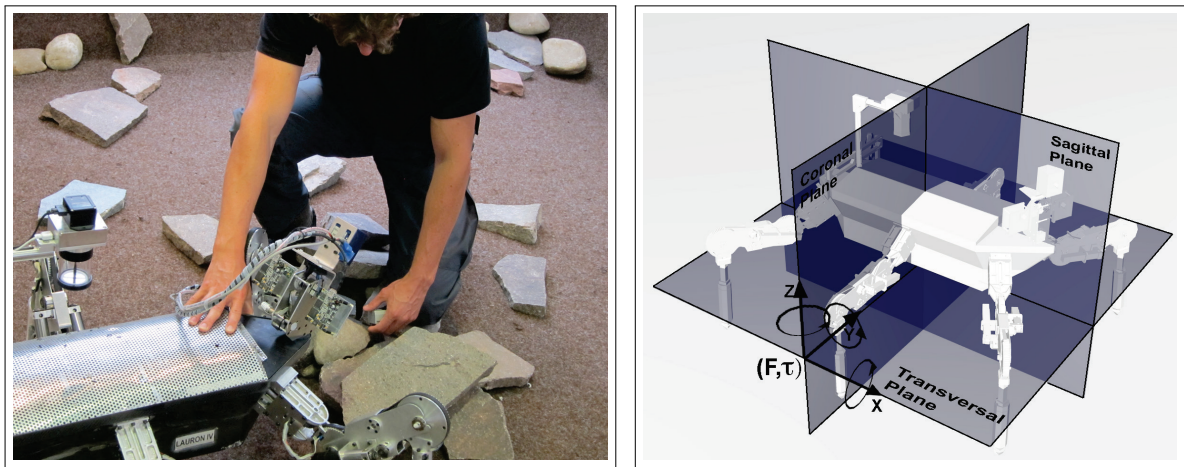


Fig. 8.4.: On the walking robot *LAURON* the components for detecting and executing *Force Commands* have been integrated (left). The whole robot acts as a force sensitive device here, measuring the forces applied to the body in three planes and around torques relative to three axis [62] (right).

Currently, only the components for classifying and executing the *Force Commands* have been implemented on *LAURON* to provide local HRI capabilities – see [62] for more details. Here the whole robot acts as a force sensitive device by measuring the force applied on its body based on the motor current of the joints. One could think of applications like letting the robot explore the environment, searching for specific objects, and then leading a human to the target object. Then the human has to interact with the robot (Fig. 8.4(left)).

As the FZI team has been selected to participate in the *DLR SpaceBot Cup* with *LAURON* (or *LAUROPE* as it is called in this context), there is the strong possibility that components of the presented control system will also be applied here. Currently, the design of the architecture is still in an early stage.

#### 8.2.5. Implementation on CityPod

The robot *CityPod*<sup>2</sup> is based on a *Segway* platform (Fig. 8.5 (left)). It can be steered by its driver while being assisted with collision avoidance functionalities. Additionally, it can navigate autonomously to a user who called it using a mobile device. The reactive and tactical parts of the control system utilize the same modules

<sup>1</sup>Developed at FZI/IDS by A. Rönnau and G. Heppner

<sup>2</sup>Developed at FZI/TKS by F.Steinhardt

as presented in this thesis (Fig. 8.5 (right)): The behaviors for avoiding static and moving obstacles as well as the predictive obstacle handling (*LookForCorners*). The *Behavior Network* is extended by a behavior to enable the robot to flow within groups of people, the behavior *Flock* [155], which would also be beneficial in crowded situations on *InBOT*.

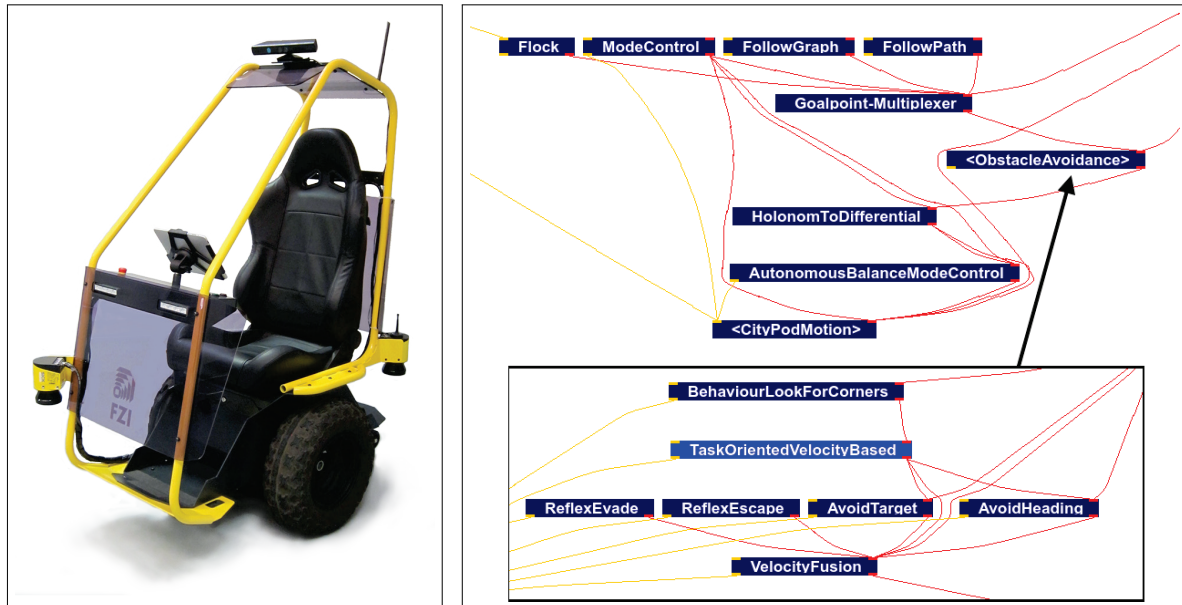


Fig. 8.5.: The differential driven robot *CityPod* is built based on a *Segway* platform (left). The driver can call the robot and is assisted with obstacle avoidance and navigation functionalities while driving. The control system incorporates reactive and tactical behaviors for avoiding collisions with static and moving obstacles. A highlight is the ability to move in dense crowds using the *Flock* behavior (right). Figures by courtesy of *CityPod*'s lead developer, Florian Steinhardt.

### 8.2.6. Implementation on *HoLLiE*

*HoLLiE*<sup>3</sup> (Fig. 8.6 (left)) is a semi-humanoid robot: an upper body with a head and two arms mounted on a wheeled platform with a holonomic drive system (a *Segway Omni*, Fig. 8.6 (middle)). It is aimed at mobile manipulation such as services close to home and – according to its full name *House of Living Labs Intelligent Escort* – at guiding services in FZI's *House of Living Labs (HoLL)*<sup>4</sup>. Currently, *HoLLiE* uses reactive behaviors for collision avoidance with static obstacles and safety (8.6 (right)), *HoLLiE*'s predecessor *IMP* also utilized behaviors for the avoidance of moving obstacles (described in [70]).

### 8.3. Discussion

In the course of this thesis a hierarchical hybrid control architecture was developed which incorporates reactive as well as deliberative behaviors and classical planners. All components work independently from each other. This combination is a main advantage of the concept: while the *reactive behaviors* grant suitable reaction times and are robust versus insufficient environmental information, the plan-based components can solve complex situations.

<sup>3</sup>Developed at FZI/IDS by A. Hermann (overall system) and J. Oberländer (navigation system)

<sup>4</sup><http://www.fzi.de/index.php/en/research/fzi-house-of-living-labs>



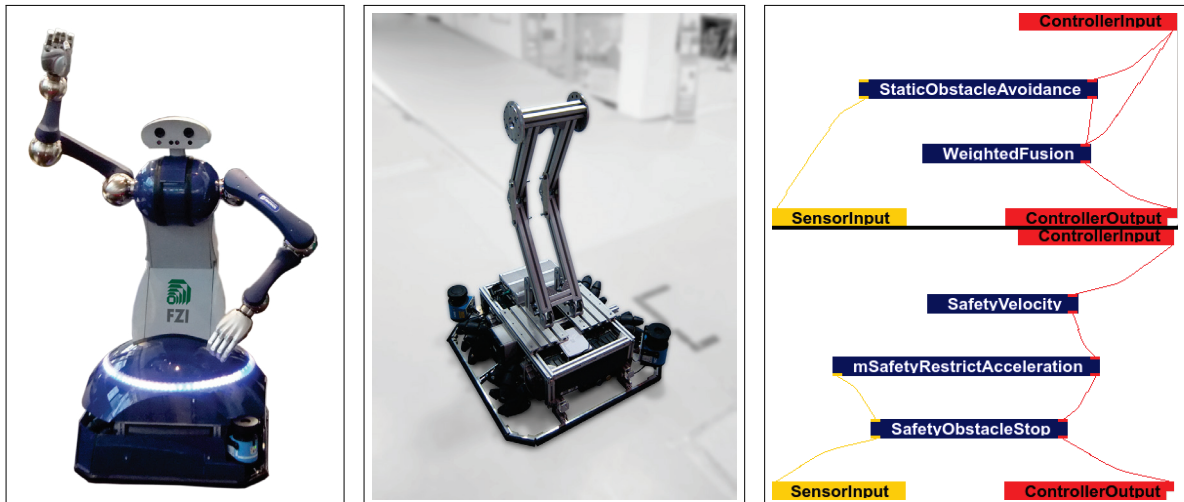


Fig. 8.6.: The semi-humanoid robot *HoLLiE* (left) is aimed at mobile manipulation tasks in the context of services close to home as well as on guiding visitors of FZI's *HoLL*. It uses reactive behaviors for collision avoidance and safety (right). Figures by courtesy of *HoLLiE*'s lead developers, Andreas Hermann (overall system) and Jan Oberländer (navigation).

The primary focus was on coping with the dynamic environment as well as cooperating with a user. There are lots of influences on the robot's actual behavior which have to be taken into account. The modularity of the *Behavior Networks* has proven valuable: integrating new behavior modules or recombining behaviors by activating or inhibiting them has been easy compared to calculating one trajectory which considers the same amount of influences.

Parts of the control system have been implemented in other robots, showing an other advantage of the modularity: as the individual layers and behaviors operate independently and have defined interfaces, they can be substituted easily, depending on the target application.

In the course of this thesis several topics have been addressed, they will be discussed individually in the next few sections.

### 8.3.1. Behavior-based control system

The control architecture developed here follows a hybrid approach, merging a *behavior-based* control system in the two lower layers and planners in the upper layer. The behavior-based part is responsible for safety, collision avoidance, navigation, and interaction with humans and other robots – which goes beyond common applications of *BBCs*. The *Behavior Network* which implements the *BBC* allows a quick and seamless integration of advanced, social and interactive behavior modules. This way advanced robot behaviors such as social behavior, guiding and following, cooperative behaviors like several assistants, and interaction/communication with humans and other robots were achieved – which again significantly extends the scope commonly found in *BBCs*.

The utilized implementation of the *Behavior-Based Control* – the *Behavior Network*, thus, a network of independent behavior modules – has one characteristic which turned out to provide simultaneously major benefits and a major drawback. The robot's overall behavior emerges from a large number behavior modules combining their share of the control self-dependent and hardly supervised.

The main benefit is the ease of integrating new functionalities compared to monolithic path or motion planners. If a new functionality is needed, the new one is implemented and encapsulated in a module. This adds its new control share to the existing ones depending on the corresponding layers' "language" (for example a velocity set-point vector). This could even be performed online (even though the current version of the MCA2 framework does not support this – the third version will) with hardly any restriction as long as steadiness is granted. This way, a significant number of functional robot behaviors can be achieved by combining the individual modules: following the user while keeping a social distance while finding a path around several obstacles in parallel to avoiding an approaching moving object. In contrast, a trajectory planner would have to calculate a common path considering all threats and influences based on a very complex and all-embracing environmental representation. From a system developers point of view, the *Behavior Network* encourages trying alternative or even new innovative behaviors, as it is not a lot of effort to integrate them.

A second benefit of the *BBC* – or more precisely the *reactive behaviors* which do not calculate a path but solely a virtual force – is that it can be applied in situations in which a trajectory planner or target-oriented methods can hardly be used at all: the user steers the robot using the *force sensitive handle*. Even in this situation, all reactive behaviors can still be used, including avoiding collisions with static and even moving obstacles. If the latter makes sense is questionable though - but it illustrates the possibilities offered by the concept.

And finally a third benefit is that the *Behavior Network* is redundant. If modules of the *BBC* fail they withdraw their control share but the remaining modules continue providing directions. The geometrical and the reactive obstacle avoidance modules do not depend on each other, and even the individual instances of the reactive ones are completely independent. The same also applies to for instance the adaptive behaviors: if the distance keeping module for the following behavior fails for some reason, the robot will not run into the user, because there are still the obstacle avoidance behaviors. And so forth.

The drawback of the concept is that the high degree of self-dependence of the modules makes it more difficult to predict the overall behavior – the robot behaves less machine-like and more "natural". If the *BBC* is parameterized properly, meaning the individual parameters for each module are set adequate, this does not pose a problem: even though some actions might seem strange at first glance, most of the time they make perfectly sense under closer examination. The real issue is that the origin of an action is not obvious and difficult to track as it results from a combination of various sources. If there is a problem or an error in one of the modules, it is laborious to even identify the responsible module because in a live system the feedback through the real world makes the individual modules interact with each other, searching for a state of equilibrium. Therefore, it is difficult to distinguish cause and effect of an observed action.

From a system designer's point of view, the presented *Behavior Network* eases integration and recombination of functionalities (even online) but from an implementation point of view optimizing and identifying errors is difficult. The *BBC* facilitates the fast integration of new functionalities, but bringing the overall behavior to perfection is cumbersome.

### 8.3.2. Dynamic environments

The primary focus of the navigation system has been on handling a dynamic environment including inaccuracies in the world model, structural changes in the environment, and moving objects.

To tackle this challenge, the control system implements a layered navigation system consisting of deliberative and reactive behaviors topped by a *topological navigation*. On its own, each layer is able to find a suitable path on the level of data abstraction corresponding to the layer. Combined, they generate a powerful navigation system able to find a path over long distances which can be nested and cluttered in detail. Moreover, this combination enables navigating without a global metric map – a *topological map* and sensor information from the robot’s current field of view are sufficient. In contrast, many methods found on “guiding robots” (except the graph-based method used by the robot *Robox*) depend on a metrically accurate and up-to-date map. This is an important characteristic as global metric maps can be outdated quickly. Especially in the shopping scenario, parked conventional shopping carts can alter the environment significantly. Suddenly a corridor seems to be a dead end or  $1m$  less wide and cannot be matched to the map anymore. Another source of confusion are semi-static objects which are moved on a day-to-day basis like special offer bins or advertisements.

Inaccuracies of the world model, the self-localization, or the task-specific databases (in the shopping scenario for example the product locations or the exact position of the topological links) are addressed by the geometrical *scene analysis* of the *tactical layer*. The target locations commanded by the *communication layer* or the *strategic layer* are interpreted as “suggestions”, only. The *tactical layer* adapts them to the actual geometry of the local environment.

### Moving obstacles

A major challenge of dynamic environments is avoiding collisions with other moving objects or humans. In a supermarket these could be customers with shopping carts who are hurrying down a corridor while being distracted. The control system unifies reactive behaviors and a plan-based approach to benefit from fast reaction times as well as from predictive mid-term planning. It embodies three main components: Firstly, a reflex moves the robot directly away from mobile obstacles which came too close or suddenly started moving, enabling *InBOT* to regain a safety distance. Secondly, a *reactive behavior* lets the robot move out of the predicted path of an approaching object. Thirdly, to solve complex situations, the behavior-based components are topped by a spatio-temporal planner which generates a safe and efficient mid-term path.

The advantage of the deliberativity of the purely planner-based solutions over reactive ones is obvious. But keeping the overall system in mind, there can be a lack of information (limited field of view, no precise and reliable map, no target location given e.g. in *Servoing Mode* or *Manual Steering Mode*) impairing a purely planner-based approach. Therefore, a combination of a planner and reactive components is used in this thesis. All components have to fit into the hierarchical concept of the control architecture. Thus, the planner does not generate a continuous trajectory but a sequence of sparse sub-goals.

Using this approach, the advantages of the planner-based approaches have been combined with the robustness regarding insufficient information of the reactive approaches.

#### 8.3.3. User interaction and control sharing

In a dynamic environment all kinds of sources exert control on the robot’s behavior. This can be parts of the control system based on sensor readings, task execution components, other robots, and so forth. As the control system is to control a service robot, the user himself will also want to exert control. An important challenge is to design user interfaces and interaction capabilities that can be used intuitively, enabling the

potential users to benefit from the robot's functionalities and to allow effective cooperative task execution. The two major sources of control shares have to be fused: the user and the control system itself. Within the *Behavior Network* the user's control input is actually considered to be just another behavior and, thus, incorporated by a fusion behavior module.

Several input modalities (force sensitive handlebar, speech, touch screen, bar code scanner, and vision-based techniques) are provided via a multimodal user interface to receive the user's control input. Five *modes of operation* (the *Autonomous*, *Guiding*, *Following*, *Servoing*, and the *Manual Steering Mode*) have been developed to orchestrate the individual behaviors and user inputs. The control on the robot is – depending on the current mode and the used modality – shared or traded between control system and user, ranging from closely coupled interaction based on physical contact up to autonomous command-based interaction. For example in the *Manual Steering Mode* the user steers the robot by exerting forces on the handlebar, having full control on the robot. In certain situations the robot can take over a share of control for itself to support the user with obstacle avoidance or other assistance features. In the vision-based case guiding or following behaviors are present with a rising degree of control that is permanently assigned to the control system.

The tests performed with users have shown that they are able to utilize the methods of controlling the robot, needing only a very brief introduction. The quantitative results of the survey performed along with the experiments show that the users have been content with the robot's abilities – even though the qualitative results show areas needing improvement, which was no real surprise as we are talking about a prototype system.

### 8.4. Open issues

In this section, some of the most prominent open issues of the current control system are summarized from a scientific point of view. Topics regarding a public roll out can be found in the next section.

**Parameter optimization of the *BBC*:** As the parameterization of the *Behavior Network* is cumbersome, this could be performed by an optimization algorithm. It is most likely not possible to express the network using a closed formula, not speaking of differentiability or even continuous differentiability. Hence, many common optimization methods aside from the evolutionary or genetic algorithms cannot be applied. These algorithms have the drawback that they need a large number of evaluations of the optimization function – which in this case can probably only be done by conducting at least simulated test runs with the system. But with strong computing hardware at hand, this still might be possible. The opposite approach would be a formal analysis of small sections of the network which can be expressed by a closed formula.

**Multi-robot path planning:** The multi-robot behaviors developed here do not perform a full-scale multi-robot path planning like for example done by R. Regele [129]. As long as only a few robots meet each other in one location the current exemplary behaviors are sufficient, but for larger fleets of robots the behaviors will not be efficient, thus, a full-scale planning process would be appropriate. If this planning would be done in a de-central manner it would be implemented in a tactical behavior module, if the planning is performed centralized a plan execution behavior would be added.

**Classification of obstacles:** Currently, the robot cannot distinguish between static obstacles which are blocking an alley, obstacles which could be removed by the robot's user, and people blocking the way

only for a very short period of time. The people could be asked to let the robot pass. The current environmental perception does not permit making this differentiation. All obstacles are treated in the same manner which can cause problems: if for example the predictive obstacle handler does not find a way in the correct direction because someone is blocking the alley, the robot will try to find a completely different way using another corridor.

**Handling of traffic jams:** Groups of people standing in an alley pose a challenge for HRI. The robot's user is able to move – or even squeeze – through a crowd, which is not possible for the larger robot. Especially while operating in the *Following Mode* the robot should be able to proactively inform the user, that it will not be able to follow if he should proceed.

**Priority of moving obstacles:** Currently the behavior modules for avoiding collisions with moving obstacles try to evade every approaching object. In crowded environments this approach quickly reaches its limits as there is not sufficient “safe” space. Thus, an assessment is needed to determine which out of the objects must be avoided at all costs and which objects will probably make evasive motions themselves. Sizes, type, and velocity of the objects might be indicators here, but also information like the direction of gaze of people – provided this information is available.

**Floating in groups of people:** When the crowds become even more dense a complete new set of motion behaviors is needed. Or at least the current ones have to be adapted to work based on velocity differences instead of distances. This way the robot would be able to float in crowds, not running into people and not making people run into it. F. Steinhardt works on extending the *Behavior Networks* with this kind of behaviors [155].

## 8.5. Roll out: from the lab to application

This section summarizes some general thoughts on issues to be addressed when rolling out this system for the public. Obviously, one could write books about this topic (and taking it literally, it surely has been done), so here only a selection of the most prominent issues will be listed.

**Peer group and purpose:** First of all, the system would have to be adapted to the purpose of the application. Shall it be used to assist elderly people? Driving slowly would be OK here. Or shall it be used mainly by impaired people – they might have completely other needs. Shall the system be used in a hardware store with wide aisles but large and heavy goods or in a small and nested grocery store? The needs of these options can be very different, if not even contrary. Without a proper definition of the purpose of the system, it can hardly be applied.

**Conflict of interests:** The application might produce a conflict between the shop owner who wants the customer to stay as long as possible in the shop and the customer himself who might just want to get over with his shopping trip – not every customer is interested in witnessing a “shopping experience”. This conflict mainly concerns the application-specific *communication layer* – but nevertheless has to be solved in advance.

**IT security:** Nowadays there is almost no day without news about hacked websites or other systems. The shopping assistant robot might handle sensitive data on the customers' shopping behavior which shall

not become public, thus, the network connections would have to be secured. Furthermore, the robot itself must not be hacked. The violator might not only get confidential information, but could sabotage the system, resulting in a lot of potential, even physical, damage. All means of access to the robot's IT hardware have to be hardened against unauthorized access.

**Safety:** Besides collision avoidance there are other safety-related issues: the system must not impair people in critical situations such as evacuations or in cases of injuries. It would be favorable if the robots would be able to assist in such situations, but at least they must be able to recognize the danger and move out of the way, especially if the central server or the WiFi were down already.

**Legal issues:** When applying such a complex system in the public, lots of legal issues arise. For example the system is equipped with sensors, and in particular cameras, rising a legal issue regarding processing and storing images of people. Furthermore, responsibilities have to be defined, in case something – or even someone – gets damaged by the robot (actively or passively).

**Engineering:** Finally, there are still several engineering issues to be tackled. Most prominently, the current system for detecting obstacles only uses planar laser range finders, thus, an extension to 3D sensors is necessary. To reduce maintenance effort, the robustness of the robotic platform has to be improved, the endurance of the accumulators increased (or the current consumption reduced), and an automatic charging station has to be set up, as had been done for the *Odete* platform in the lab. The robotic platform itself should be re-engineered, as it is too large (as can be seen on some of the pictures from user tests), and at least the height of the touch screen must be adjustable. The robot *InBOT-2* (see Sec. B.1) which is currently under development by the author will be a step in this direction.

## A. Interdependent Work, List of Publications, and Student's Theses

During the development of the control system described in this thesis, several scientific contributions on already finished components have – according to the professors' and faculty's wishes – been published prior to the publication of the completed thesis itself in order to facilitate the evaluation by the community. These are listed below.

This thesis is considered to be the mainline of the scientific work. For the sake of readability self-references and self-quotations are mostly omitted. Thus, information taken from other texts by the author are not explicitly highlighted as far as the work has been done in the course of this thesis. Self-references are given to refer to further information which did not fit into this thesis, only.

Assembling the robots and writing the thousands of lines of code which implement the designed architecture, behavior modules, and the auxiliary components would not have been possible without assistance. Thus, complementary student theses have been supervised in the course of the presented work, based on the concepts of this thesis. Accordingly, there are strong interdependencies in the contents. A complete list can be found below.

While working on this thesis, the author has taken part in the project “CommRob” (EC FP6 STREP – IST-045441) which partially focused on similar topics. When delays with the *CommRob Demonstrator (CoRoD)* rendered the project without a robotic platform for development, test, and demonstration, the robot *InBOT* was placed at the disposal of the CommRob project. As result, the work done by the author in the course of “CommRob”, and in particular the reports written by him for the work packages “Advanced Robot Behavior and Navigation” and “Robot Integration and Test” which have been his responsibility, and some chapters of this thesis are interdependent.

During CommRob, several components of the software developed by other groups have been integrated in the course of this thesis to allow for evaluation setups. This counts especially for M. Devy's group at LAAS CNRS and H. Kaindl's group at TU Vienna. With the latter, especially with D. Ertl, as well as with H. Hüttenrauch, A. Green, and C. Bogdan from KTH Stockholm, several collaborative activities like user experiments and integration sessions have been conducted, which resulted in collaborative publications on Human Robot Interaction. Accordingly, when describing the components in this thesis, it is indicated if the particular component was developed by persons other than the author of this thesis himself.





## List of Publications

- [GRG<sup>+</sup>11] Michael Goeller, Arne Roennau, Anton Gorbunov, Georg Heppner, and Rüdiger Dillmann. Pushing Around a Robot: Force-Based Manual Control of the Six-Legged Walking Robot LAURON. *In: Proc. IEEE International Conference on Robotics and Biomimetics (RoBio11), Phuket, Thailand, 2011.*
- [GSK<sup>+</sup>10a] Michael Goeller, Florian Steinhardt, Thilo Kerscher, Rüdiger Dillmann, Michel Devy, Thierry Germa, and Frederic Lerasle. Sharing of control between an interactive shopping robot and its user in collaborative tasks. *In: Proc. 19th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Viareggio, Italy, pages 626–631, September 2010.*
- [GSK<sup>+</sup>10b] Michael Goeller, Florian Steinhardt, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. Proactive avoidance of moving obstacles for a service robot utilizing a behavior-based control. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Taipei, Taiwan, Oct., pages 5984–5989, 2010.*
- [GSK<sup>+</sup>09] Michael Goeller, Florian Steinhardt, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. Robust navigation system based on RFID transponder barriers for the interactive behavior-operated shopping trolley (InBOT). *Industrial Robot, Emerald Group Publishing Limited, 36(4):377–388, 2009.*
- [GBK<sup>+</sup>09] Michael Goeller, Malco Bluemel, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. Can the Modeling of Pedestrian Movements Improve Robot Behaviors? *In: Proc. Workshop on Improving Human-Robot Communication with Mixed-Initiative and Context-Awareness, 18th IEEE International Symposium on Robot and Human Interactive Communication (RoMan), Toyama, Japan, Oct., 2009.*
- [GKZ<sup>+</sup>09a] Michael Goeller, Thilo Kerscher, Marco Ziegenmeyer, Arne Roennau, J.Marius Zoellner, and Rüdiger Dillmann. Haptic control for the interactive behavior operated shopping trolley InBOT. *In: Proc. New Frontiers in Human-Robot Interaction, Convention Artificial Intelligence and Simulation of Behaviour (AISB), Edinburgh (UK), 2009.*
- [GKZ<sup>+</sup>09b] Michael Goeller, Thilo Kerscher, J.Marius Zoellner, Rüdiger Dillmann, Michel Devy, Thierry Germa, and Frederic Lerasle. Setup and control architecture for an interactive Shopping Cart in human all day environments. *In: Proc. International Conference on Advanced Robotics (ICAR), 2009.*
- [GKZD09] Michael Goeller, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. Obstacle Handling of the Holonomic-driven Interactive Behavior-operated Shopping Trolley InBOT. *In: Proc. IEEE 7th International Workshop on Robot Motion and Control (RoMoCo09), Czerniejewo, Poland, 2009.*
- [GSKZ09] Michael Goeller, Florian Steinhardt, Thilo Kerscher, and J.Marius Zoellner. Reactive Avoidance of Dynamic Obstacles using the Behavior Network of the Interactive Behavior-Operated Shopping Trolley InBOT. *In: Proc. International Conference on Climbing and Walking Robots (CLAWAR), Istanbul, Turkey, September 2009.*
- [GKZD08] Michael Goeller, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. Behavior Network Control for a Holonomic Mobile Robot in Realistic Environments. *In: Proc. International Conference on Climbing and Walking Robots (CLAWAR), Coimbra, Portugal, September 2008.*
- [GSK<sup>+</sup>08] Michael Goeller, Florian Steinhardt, Thilo Kerscher, J.Marius Zoellner, and Rüdiger Dillmann. RFID Transponder Barriers as Artificial Landmarks for the Semantic Navigation of Autonomous Robots. *In: Proc. 11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), Coimbra, Portugal, September 2008.*
- [BEH<sup>+</sup>11] Cristian Bogdan, Dominik Ertl, Helge Huettenrauch, Michael Goeller, Anders Green, Kerstin S Eklundh, Jürgen Falb, and Herman Kaindl. Evaluation of Robot Body Movements Supporting Communication: Towards HRI on the Move. *New Frontiers in Human-Robot Interaction, pages 185–210, 2011.*
- [BEG<sup>+</sup>10] Cristian Bogdan, Dominik Ertl, Michael Goeller, Anders Green, and Kerstin S Eklundh. Towards HRI on the Move with Mixed Initiative. *In: Proc. 2nd International Symposium on New Frontiers in Human-Robot Interaction, Convention for the Study of Artificial Intelligence and Simulation of Behaviour (AISB), Leicester (UK), March, 2010.*
- [HCG<sup>+</sup>10] Helge Huettenrauch, Bogdan Cristian, Anders Green, Kerstin S Eklundh, Dominik Ertl, Jürgen Falb, Herman Kaindl, and Michael Goeller. Evaluation of Robot Body Movements Supporting Communication. *In: Proc. 2010 Convention Artificial Intelligence and Simulation of Behaviour (AISB2010), Leicester, UK, pages 1–8, 2010.*
- [BG09] Cristian Bogdan and Michael Goeller. Towards a Framework for Design and Evaluation of Mixed Initiative Systems: Considering Movement as a Modality. *In: Proc. Workshop on Improving Human-Robot Communication with Mixed-Initiative and Context-Awareness, 18th IEEE International Symposium on Robot and Human Interactive Communication (RoMan), Toyama, Japan, Oct., 2009.*
- [KGZ<sup>+</sup>09] Thilo Kerscher, Michael Goeller, Marco Ziegenmeyer, J.Marius Zoellner, and Rüdiger Dillmann. Intuitive control for the mobile Service Robot InBOT using haptic interaction. *In: Proc. International Conference on Advanced Robotics (ICAR), Munich, Germany, 2009.*
- [RGD<sup>+</sup>09] Younes Raoui, Michael Goeller, Michel Devy, Thilo Kerscher, J.Marius Zoellner, Rüdiger Dillmann, and A Coustou. RFID-based topological and metrical self-localization in a structured environment. *In: Proc. International Conference on Advanced Robotics (ICAR), Munich, Germany, 2009.*



## List of Students' Theses

- [Bil10] Christian Billet, *Internship Semester Report Winter Term 2009/2010*, Internship report, Advisor: M. Goeller, Supervisor: Norbert Skricka, Faculty for Mechanical Engineering and Mechatronics, University of Applied Sciences Karlsruhe, 2010.
- [Boe10] Daniel Boettinger, *Internship Semester Report Winter Term 2009/2010*, Internship report, Advisor: M. Goeller, Supervisor: Norbert Skricka, Faculty for Mechanical Engineering and Mechatronics, University of Applied Sciences Karlsruhe, 2010.
- [Bre09] Kathrin Breiner, *Analyse des Marktpotentials für Einsatz und Entwicklung technischer Einkaufshilfen*, Bachelor thesis, Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Information Engineering and Management, University of Karlsruhe (TH), 2009.
- [Gab09] Stefan Gabriel, *Technische Einkaufshilfen - Herausforderungen, Erwartungen der Kunden und bisherige Entwicklungen*, Seminal thesis, Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Information Engineering and Management, University of Karlsruhe (TH), 2009.
- [Gob07] Vadim Gobulev, *Positionskorrektur mittels künstlicher Infrarot-Landmarken am Beispiel des North Star Systems*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2007.
- [Gor12] Anton Gorbunov, *Intelligent Shopping Assistant: The Next Shopping Experience*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, Karlsruhe Institut for Technology (KIT), 2012.
- [Hei09] Thomas Heinkel, *Entwicklung eines Lokalisierungskonzeptes für beliebige mobile Plattformen basierend auf spärlich verteilten RFID Landmarken*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2009.
- [Ö10] Burc Özüpek, *Erweiterte Realität zur Visualisierung von Roboterverhalten basierend auf MCA*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2010.
- [Ost10] Roland Ostrowski, *Praxissemesterbericht (Internship Semester Report)*, Internship report, Advisor: M. Goeller, Supervisor: Norbert Skricka, Faculty for Mechanical Engineering and Mechatronics, University of Applied Sciences Karlsruhe, 2010.
- [Rit10a] Hugo Ritzkowski, *Entwicklung von Algorithmen zur Bewältigung spezieller Situationen bei der Navigation mobiler Roboter*, Bachelor thesis, Advisor: M. Goeller, Supervision: Klaus Wüst, Peter Löffler, Faculty for Computer Science Fachhochschule Giessen Friedberg, 2010.
- [Rit10b] Hugo Ritzkowski, *Praktikumsbericht (Internship Report)*, Internship report, Advisor: M. Goeller, Supervisor: Klaus Wüst, Faculty for Computer Science, Fachhochschule Giessen Friedberg, 2010.
- [Ste08] Florian Steinhardt, *RFID-Transponder als künstliche Landmarken für die topologische Navigation einer mobilen Plattform*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2008.
- [Ste09a] Fabian Stehle, *Entwicklung eines flexiblen hierarchischen Planers für Aufgabenkoordination und topologische Navigation*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2009.
- [Ste09b] Florian Steinhardt, *Entwicklung einer proaktiven verhaltensbasierten Kollisionsvermeidung mit dynamischen Hindernissen*, Diploma thesis, Advisor: M. Goeller, Supervision: R. Dillmann and U. Hanebeck, Faculty for Computer Science, University of Karlsruhe (TH), 2009.
- [Ste11] Fabian Stehle, *Fusion verteilter Sensoren zur Personenverfolgung für adaptive Verhalten von Service-Robotern*, Diploma thesis, Advisor: M. Goeller, Supervision: R. Dillmann and U. Hanebeck, Faculty for Computer Science, Karlsruhe Institut of Technology (KIT), 2011.
- [Zie08] Artur Zientara, *Simulation der Kinematik und Odometrie für omnidirektionale Fahrzeuge*, Studienarbeit (mid-study thesis), Advisor: M. Goeller, Supervisor: R. Dillmann, Faculty for Computer Science, University of Karlsruhe (TH), 2008.



## B. The Interactive Behavior-Operated Trolley (InBOT), InBOT-2, and ETrolley

This chapter introduces three robotic platforms: *InBOT*, *InBOT-2*, and *ETrolley*, along with several hardware components used on these systems: the concept for self-localization based on a pair of 2D measurement wheels and RFID barriers as well as the *force sensitive handle*.

The robot system *InBOT* has been designed as development platform for the control system and to facilitate testing the results in the context of the chosen scenario of application: shopping in a supermarket.

The enhanced conventional trolley *ETrolley* has been designed as a system to support user studies and to be tested in a real store in an early stage. It consists of an ordinary shopping trolley with the passive components of *InBOT*, thus, the system for self localization, the PCs, and the touch screen interface.

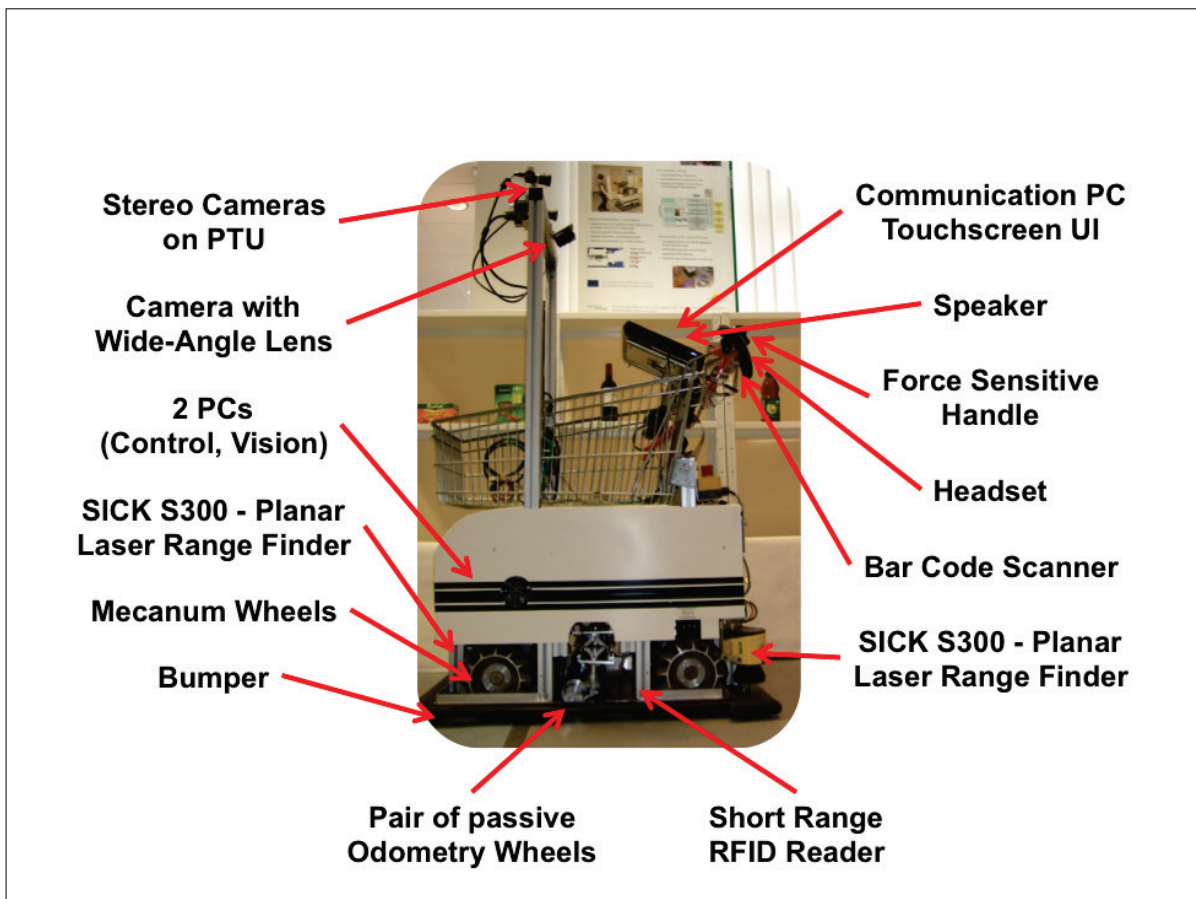


Fig. B.1.: *InBOT*'s equipment.

Several challenges arise from the implications of the supermarket scenario for the design of the robot or the choice of equipment for the robot. Figure B.1 given an overview on *InBOT*'s equipment and Figure B.2 sketches the integration of the hardware components.

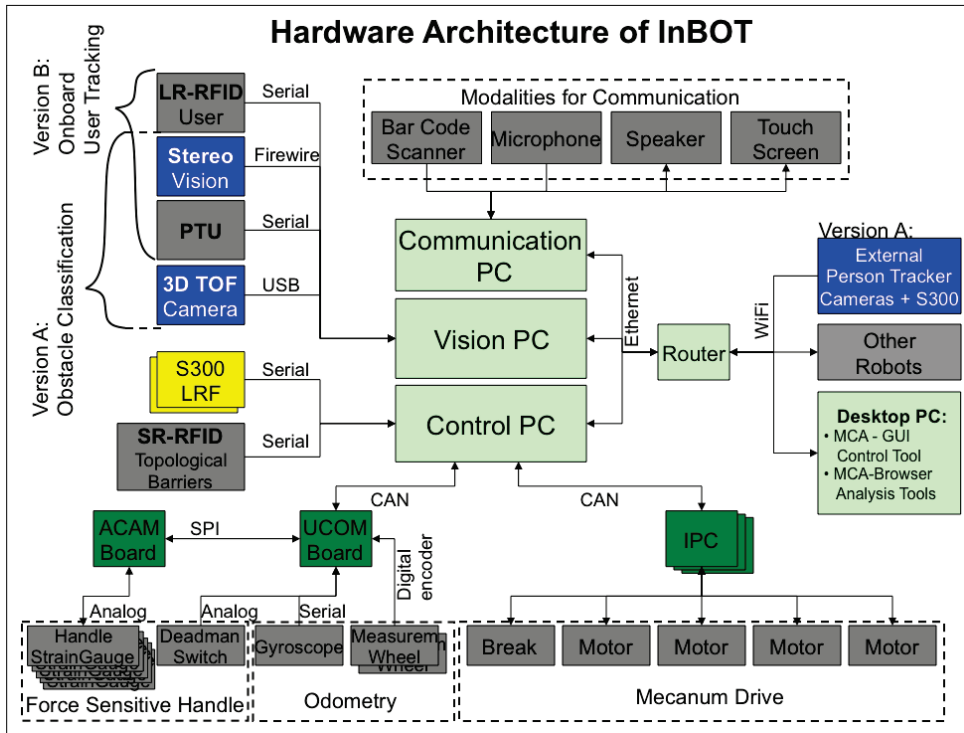


Fig. B.2.: Hardware architecture of *InBOT*.

The root challenge regarding the hardware posed by the scenarios is the need to combine a high degree of maneuverability of the drive system with the ability to transport goods in a similar amount as a conventional shopping cart. This results in a causal chain of implications: first of all a small circular robot platform with a differential drive cannot be used. Instead, a larger rectangular platform has to be used to support a shopping basket of a suitable size. The motors have to provide for the power to accelerate 50kg of goods, resulting in a high power consumption and larger accumulators. To ensure the needed maneuverability in narrow and cluttered corridors, an omni-directional drive is needed. On *InBOT* a holonomic *Mecanum Drive* is used. The drawback is that the drive system is subject to a significant amount of slipping which decreases the quality of the odometry information from the driven wheels. Therefore special passive measurement wheels had to be designed to acquire precise odometry data. When being used with an holonomic drive, these have to have at least 2 mechanical degrees of freedom (DOF) and an overall 3 DOF have to be measured. Additionally, the use of omni-directional drive systems means a full 360deg environmental perception with a suitable precision is needed to be able to avoid obstacles while driving sideways or turning on the spot with a non-circular robot. Here two SICK S300 laser range finder, each with a 270-degree field of view have been used at opposite corners of the robot, providing for a full 360 degree field of view.

One of the requirement demanded the presence of a force sensitive input device for the *Manual Steering Mode*. Thus, according to the scenario, a force sensitive handlebar has been designed.

### B.1. The robot *InBOT2*

*InBOT* is a prototype, thus, a large system with a low degree of integration. Currently, the robot *InBOT2* is being designed with two major requirements:

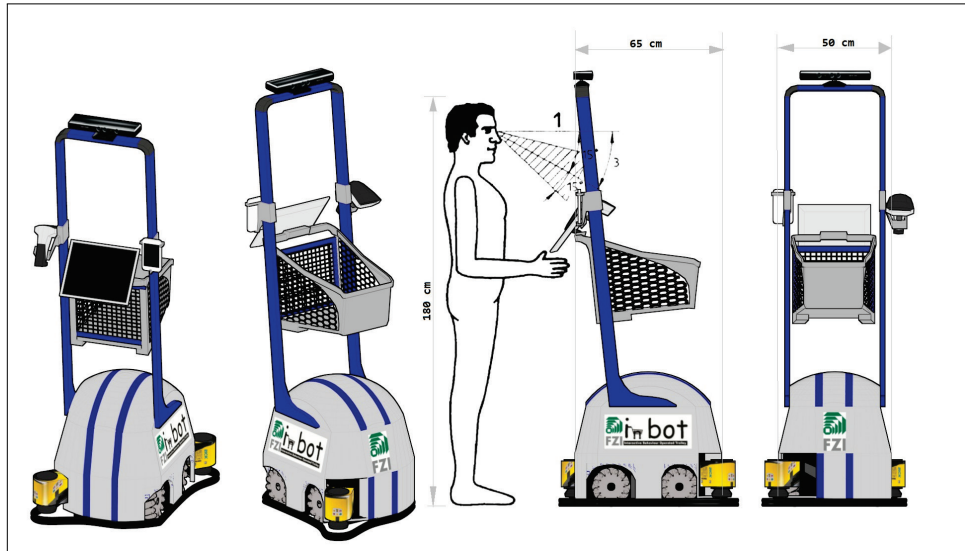


Fig. B.3.: *InBOT2*: Design sketch of the robot from different points of view, alongside with a human silhouette (image source: [33]) for comparison.



Fig. B.4.: *InBOT2* – Left: CAD data. The minimal footprint of the robot is dominated by the size of the two S300 (yellow) and the motors including their gearboxes. The upper levels of the robot's body are designed in a drawer concept, reducing the unused space between the components while providing for easy access to the components. The two major components to be taken into account are the embedded PCs – the green boxes illustrate the bounding boxes including plugs. Right: Current status of the system.

- Smaller size: The robot shall be usable in smaller supermarkets, hence, the robot's footprint shall be reduced to half the current size. This can be achieved by rising the level of integration while a drawer-concept improves the accessibility of the individual components. The size of the basket has to be reduced significantly too, as it is a major factor in the robots footprint. The height of the robot is fixed: the height of the basket shall still be same same like on ordinary shopping carts.
- Better maintainability: Learning from the issues of the first *InBOT*, the robustness of the robot shall be increased. This refers mostly to the electronics system. As the setup is more or less finalized the cabling can be fixed, plugs can be removed, and so forth.

Figure B.3 shows a design sketch of the system from different points of view. For a comparison of the robot's size, a human silhouette is added. Almost all externally visible components but the S300 have been reduced in size. Figure B.4 (left) shows a CAD drawing of the interior of the robot. The size of the basis is defined by the two S300 as well as by the motors and gearboxes. The remaining components fit into the drawer-system on top. Finally, B.4 (right) shows the current status of the system. The most time consuming steps have been taken and first driving tests have been successful.

## B.2. ETrolley

The system *ETrolley* was designed to facilitate early tests, user studies, and test runs in a real store. *ETrolley* is actually not a full-scale robot as it lacks the active components. Nevertheless, it runs the complete navigation system of *InBOT* for the purpose of providing navigation assistance for the user. For this purpose it is equipped with the complete system for local and global self-localization and with the communication PC housing the user interface. Figure B.5 (top) shows the individual hardware components of the system and Figure B.5 (bottom) sketches the hardware architecture.

As there have been several purposes for the system, several instances have been set up. There is a version using the *force sensitive handle* for analyzing how humans actually steer a shopping cart (Fig. B.6 (left)). It has no screen-based user interface as the users shall not be distracted. Another version has been designed to act as (passive) shopping assistant, offering shopping list management and navigation assistance (Fig. B.5). And finally there has been a version which focused on easy and modular assembly which has been used in a real supermarket. According to the wishes of the shops owner, the robotic functionalities should be disguised as far as possible (Fig. B.6 (right)).



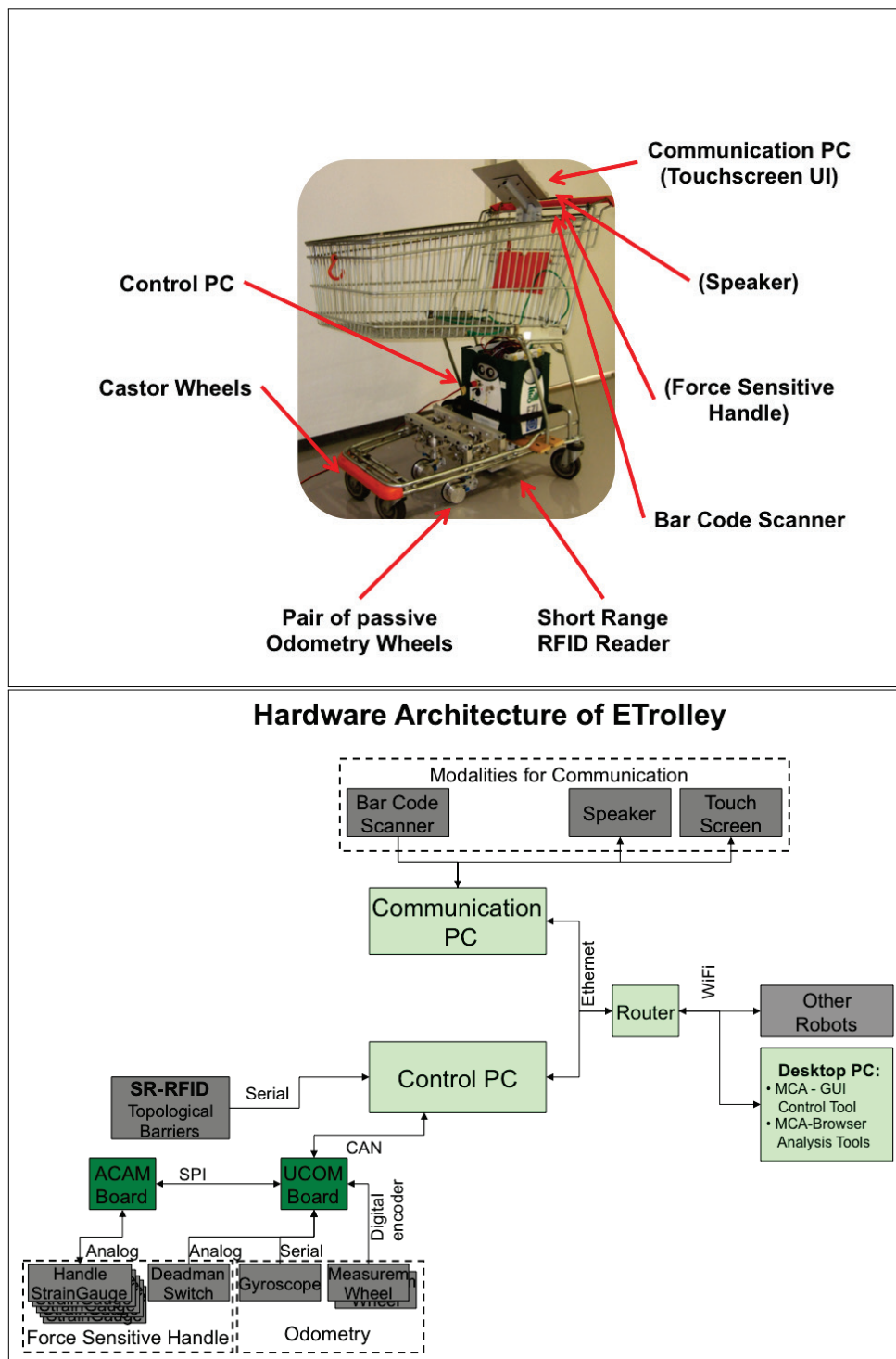


Fig. B.5.: Hardware architecture of *ETrolley*: The system *ETrolley* incorporates all the components of *InBOT* which do not depend on the drive system, e.g. the shopping list management and navigation assistance. It is equipped with a touch screen, a PC hidden in a crate, the measurement wheels for odometry, and the RFID scanner.



Fig. B.6.: Several versions of *ETrolley* have been set up: the one shown in Figure B.5, one for tests in the FZI labs including the *force sensitive handle* (left) and one reduced version for tests in a ICA Maxi supermarket in Stockholm (right, thanks to Helge Hüttenrauch for this opportunity).

### B.3. Electrical layout

The electronic layout of *InBOT* consists of a large number of circuits and components which can not be presented here in detail. Figure B.7 shows the top level of the layout. The main components are listed below:

- Accumulators (24V (2x12V), 105Ah, 35kg each)
- DC/DC converter to 12V and 5V
- 4 Motor driver Metronix DIS-2 48/10 (24-48V, 10A continuous), level converter circuits for the motors' Hall sensors
- 4 Motors Beldrive AEC 140/36 (24V, 315 U/min, 36Nm, 14A, 8kg – obviously, these motors are overpowered for the specific robot)
- 2 UCOM boards (handle, odometry sensors)
- Force sensitive handle (strain gauges, ACAM boards)
- Safety circuit (emergency stop buttons, bumpers)
- Several protection circuits (over voltage, under voltage, over current)
- Break control circuit (off, on, automatic)
- Gyroscope with circuit for time delay on switch-on



## B.4. Self-localization

This section introduces the system implemented for the self-localization of the robot (see [Hei09] for a detailed description). As defined by requirement R3 in the introduction, no detailed global map is available. Hence, popular scan-map-matching methods for the self-localization cannot be applied. Furthermore, it has to be expected that the robot is often surrounded by large numbers of people, impairing a continuous localization based on wall-mounted landmarks. Techniques would be needed which are independent from crowded corridors, colorful or low-contrast background and changing light conditions. Additionally, the landmarks would have to be sparsely distributed to keep the integration effort as low as possible.

Resulting from these challenges, a two level approach has been used in this concept: a local component tracking the motion of the robot and a global component based on sparse landmarks to correct the incremental error of the local component. The global component is also used to perform the topological self-localization.

### B.4.1. Odometry for local self-localization

The local self-localization is responsible for tracking the local motion of the robot as precisely as possible. It uses two components: The first one uses incremental encoders to track the robot's relative pose and position changes, the second one is an optional gyroscope to track the orientation changes more precisely.

**Gyroscope:** A gyroscope<sup>1</sup> is used on *InBOT* in order to track the orientation of the robot. After calculating and subtracting the drift resulting from the degree of latitude of the FZI labs it has a drift of about 3 degree/hour and an error of lower than 0.15 degree per revolution, therefore lower than 0.04%.

**Measurement Wheels** A pair of 2D measurement wheels (see Fig. B.8 and Section B.5 for construction details) has been constructed to measure the linear – therefore the lateral and longitudinal – movements of the holonomic robot. The orientation of the robot is measured too, even if more prone to errors. If a gyroscope is present for more accurate orientation measurements, this term can be used to perform corrections, if no gyroscope is present it provides the orientation value. The wheels each measure the driven distance by accumulating the covered angle as well as the orientation of the wheel orthogonal to the vertical axis. The orientation of the two wheels (1 DOF each) is correlated, but the remaining 3 DOF are sufficient to calculate the 3D planar movement of the robot. The encoders utilized have a resolution of 0.09 degree. The resolution of the rolling wheel on the ground is based on its diameter and is about 0.06mm. The final longitudinal error of the rolling wheel has a value of about 2mm per meter, therefore 0.2%. The error of the complete system of two wheels with four encoders after performing movements in 3 DOF is in X-direction 2.3mm/m, Y-direction 4.6mm/m, therefore overall 5.1mm/m or 0.5%.

**Evaluation** Following the ideas of the University of Michigan Benchmark Test (UMBmark [23]) the self-localization of the system was tested: a rectangle was driven clockwise and counter-clockwise and measured separately to identify systematic errors. The evaluation was performed on two systems: on the robot *InBOT* using two measurement wheels and a gyroscope and on *ETrolley* using two measurement wheels only. Figure B.9 shows one example out of the performed tests. The results summarize as follows:

---

<sup>1</sup>LITEF micro-force6



Fig. B.8.: The pair 2D odometry wheels mounted between the front and the back wheels of the mecanum drive track the path of the robot (see Fig. B.14 for a larger picture).

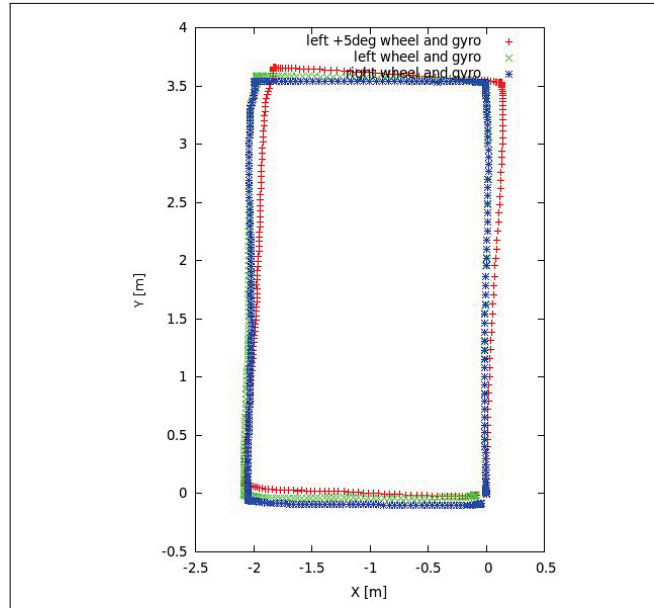


Fig. B.9.: This figure shows an exemplary test run for the measurement wheels and the gyroscope: two wheels were used in combination with the gyroscope (blue and green line). In one case the wheel was on purpose calibrated badly (red line).

**InBOT – measurement wheels plus gyroscope:** A rectangle of 3.6 x 2.0 m was driven 10 times in both directions. At the endpoint the average error was 0.09 m (0.8%) with a variance of 0.00137. The maximum error was 0.16 m or 1.4%.

**ETrolley – measurement wheels only:** The same test was performed with the same measurement wheels but without the gyroscope. A rectangle of 3.6 x 2.0 m was driven 10 times in both directions. At the endpoint the average error was 0.14 m (1.2%) with a variance of 0.0026. The maximum error was 0.31 m or 3.47%.

### B.4.2. Global self-localization by landmarks

As commonly known, the accumulated position tracking has to be corrected due to also accumulating errors. Therefore some kind of global position correction is needed. In outdoor scenarios usually GPS is applied, in indoor scenarios either WiFi localization for rough estimations or landmark-based concepts for precise calculations are used. The concept presented here utilizes precise global position corrections by detecting RFID tags mounted on the floor. Additional information and results can be found in [64], [128] and in particular in the mid-study thesis by F. Steinhardt [Ste08].

**RFID barriers as artificial landmarks:** Many landmark-based systems for self-localization share a common disadvantage compared to a ground-mounted short-range Radio Frequency Identification (RFID) system. The landmarks can be easily occluded by people or objects located in the line of sight. Wall-mounted long-range RFID-based systems can be blocked as well, for example if too many humans or metallic objects are located between transponder and reader. To avoid these occlusions and to get more precise position

information the presented concept proposes to use ground-mounted RFID transponders as landmarks as illustrated in Figure B.10. This way an occlusion of the landmarks is hardly possible.

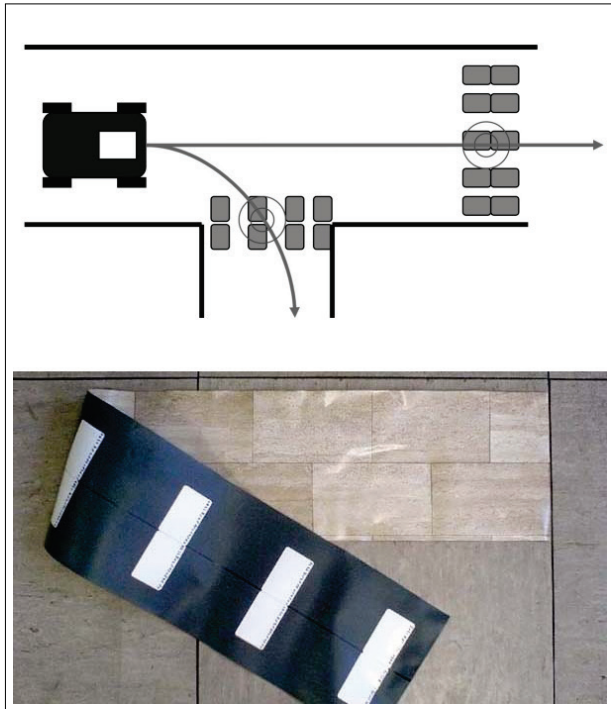


Fig. B.10.: Top: Placement of RFID barriers to divide a corridor in areas. Bottom: Picture of an RFID barrier with pairs of RFID transponders glued beneath a piece of PVC flooring.

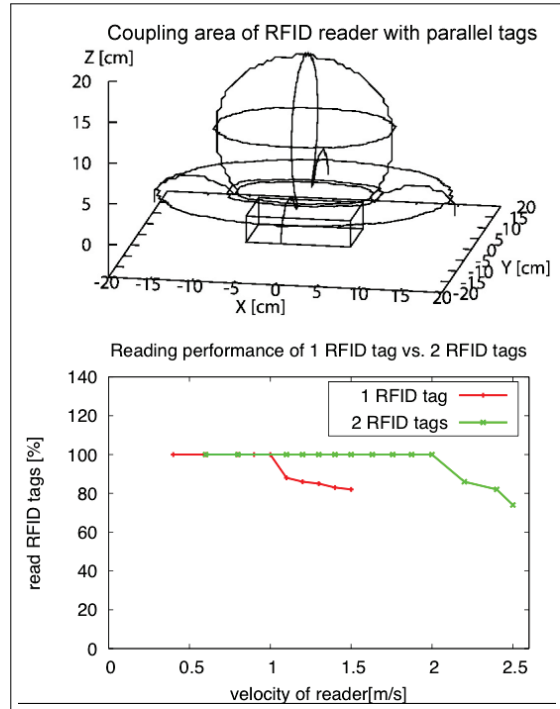


Fig. B.11.: The largest diameter of the coupling area is found in a distance of 2 cm from the reader (top). If using double-tags in the barriers the barriers are detected very reliably up to velocities of 2m/s (bottom).

The RFID transponders are organized in groups that form barriers. The tags shall be placed densely along the borders of the topological areas (Fig. B.10). This way the precision of densely distributed tags can be achieved without having to lay out complete carpets of tags. The robot knows all IDs of the tags contained in a barrier as well as the relative positions of tags within the barriers. At the moment of driving past a barrier the robot recognizes that it just entered a new area. By doubling the tags, as depicted, it even knows in which direction it has moved past the barrier. Due to the well-defined assignment of tags, barriers, and areas, this information is redundant but can be used to solve errors or hijacking problems.

To identify a proper height for assembling the RFID reader on the robot, the reader's coupling area was measured under the condition of parallel transponder and reader. As shown in Figure B.11 (top), a distance of 2cm between reader and barrier has proven optimal. Here the coupling area, consisting of a combination of the main and a side area, has approx. 35cm in diameter. The larger the coupling area in diameter the faster the robot can move past the barrier without missing it and the sparse can the tags be placed. The distance between each pair of double tags can measure up to 20cm. Additionally, the low distance of 2cm makes disturbances between reader and barrier very unlikely. Using this setup the robot was able to detect all barriers in the tests up to velocities of approx. 2m/s when double tags have been used. The reading performance with a single tag is significantly inferior (Fig. B.11 – bottom).

**Global self-localization using RFID barriers:** The relative position of the barrier, the position of the tags inside the barrier, and the time stamp when the first tag was read are known. Then the two components of the robot's location can be measured by two different methods (Fig. B.12):

- The position of the robot along the barrier: The robot's position along the barrier can be estimated by averaging over the position of the read tags. The measured transponders are compared with their position in the barrier. Because their position is known, the position of the reader can be estimated with an accuracy of plus/minus 10cm (equal to the distance between the tags).
- The position orthogonal to the barrier is measured by comparing the time stamp when the first tag was detected and the last tag was not detected any more with the diameter of the coupling area of the reader and the current velocity. This way the position can be estimated with an accuracy of not worse than plus/minus 10cm.

Figure B.13 shows a path driven by the robot. Shortly after passing the RFID barriers the position of the robot is corrected slightly.

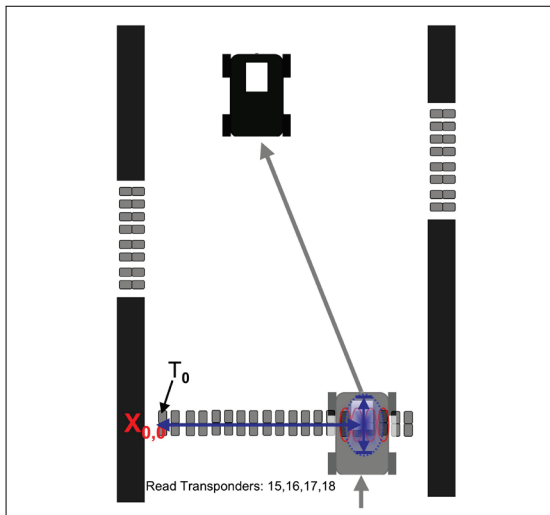


Fig. B.12.: Global self-localization using barriers of RFID tags: The tags are placed at the borders of the topological areas in form of lines. Double lines are used to be able to identify the driving direction. The position of the barriers' "low" side (tag No. 0 ( $T_0$ )) relative to the area is known. Additionally, the position of each tag in a barrier is known relative to  $T_0$ . At the very moment when the robot's RFID reader (white box) crosses the barrier, it reads a set of tags. Knowing the characteristic of the reader, the robot's exact position can be estimated.

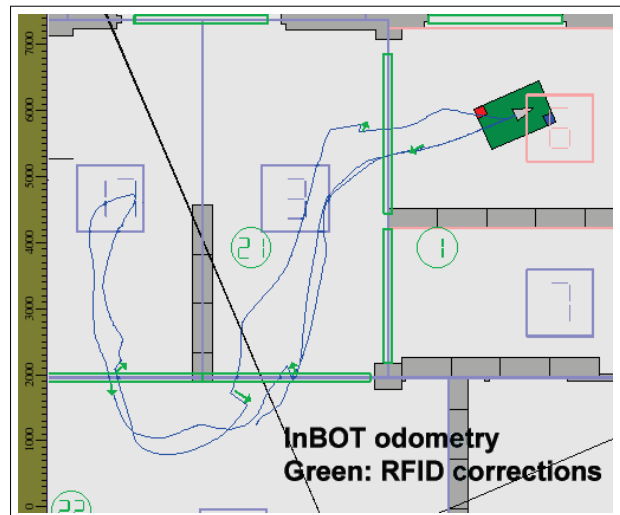


Fig. B.13.: This figure shows a screenshot from the MCAGUI after driving *InBOT* manually about 20 meters. The blue line is the path taken by the robot, the green rectangles represent the borders of the topological areas outfitted with RFID barriers. The small green arrows indicate the absolute position correction of *InBOT* after passing a barrier.

## B.5. 2D Odometry wheels

As has been mentioned in the preceding section, a pair of 2D wheels for measuring the robot's odometry has been designed (Fig. B.14). The holonomic drive has three degrees of freedom (DOF), so at least accumulated

3 DOF have to be measured by the odometry wheels – as long as the individual DOF are not correlated. Furthermore, each of the odometry wheels has to be able to follow 3 DOF motions to avoid slipping.

The designed odometry wheels each have two mechanical DOF: the wheel rolling on the floor and the rotary feedthrough connecting the wheel to the robot's body (see Figure B.15 (left), indicated by the green arrows). These two DOF are outfitted with incremental encoders to allow for measuring the distance driven on the floor and the direction change. The third DOF is given by the fact that (in an ideal case) the wheel touches the ground in one point, only. The data of the four encoders (two for each of the two odometry wheels) is handed over to the UCOM board via digital I/Os. The wheels use a spring-mounted suspension to provide for continuous ground contact in case of uneven floor (and to protect the wheels from damage when driving over bumps).

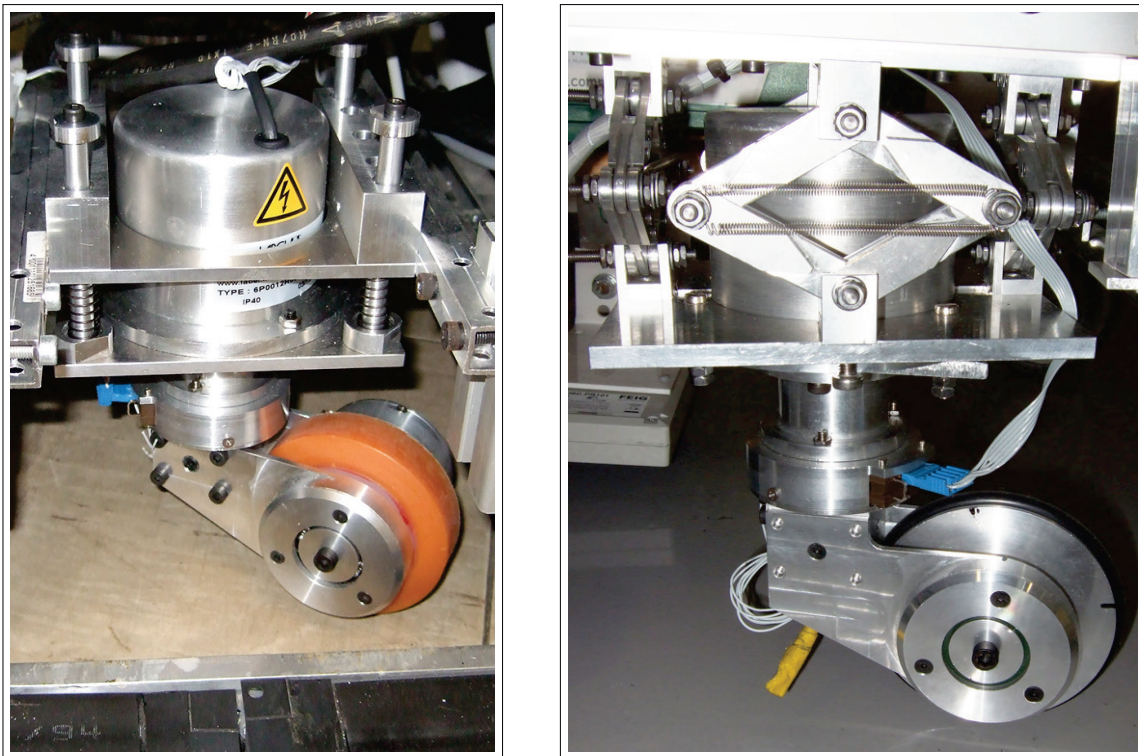


Fig. B.14.: The 2D odometry wheel tracks the path its center point has traveled. It measures the distance which it has rolled off and the direction of the rolling motion relative to the robot using wheel encoders. To smooth uneven floors it is mounted on a spring mechanism.

Left: first version with sleeve bearing and 2cm wide running surface.

Right: second version with alternative suspension and torus-shaped running surface.

Two versions have been designed. The main requirement for the first version has been a quick development process as the first prototypes have been needed for software tests. The second version fixed mainly two issues identified (Fig. B.15, see the internship report of C. Billet [Bil10] for a thorough description):

- The suspension mechanism could jam because the force applied when driving past a bump in the floor is not applied in parallel to the direction of the linear sleeve bearing's DOF. The reason is that a castor is used for the wheel and thus the point of contact between wheel and floor is located on a circumference around the axis of the bearing. The design of the second suspension is free of such jams.



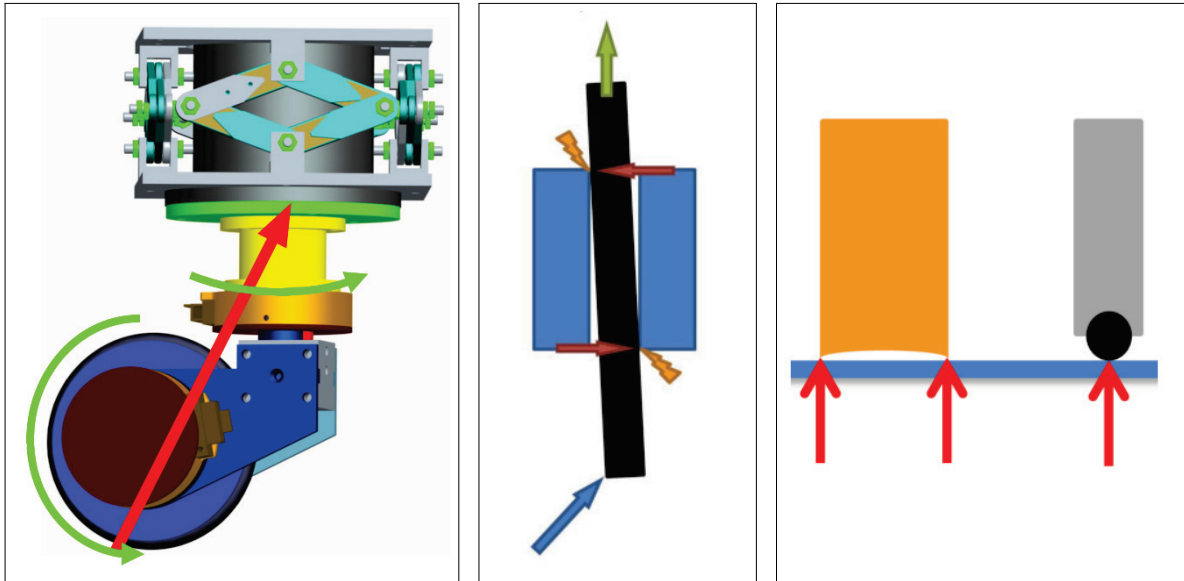


Fig. B.15.: Issues of the first version of the odometry wheel [Bil10]. Left: the force vector (red arrow) points from the ground contact point towards the suspension and, thus, is not parallel to the DOF of the suspension. The result is the possibility of high friction and jams (middle) when a linear sleeve bearing is used. Right: the running surface of the first version's wheels is wide and flat – actually it has even been slightly concave. The new running surface uses an O-ring, providing for a well defined point of contact.

- The main source of inaccuracy of the first version has been the wide and flat running surface of the wheel, thus, a distinct point of contact between floor and wheel is not granted. The second design uses a torus-shaped running surface, providing for a distinct point of contact.

## B.6. Force sensitive handlebar

The most intuitive method of using a robot trolley is probably the interaction based on physical contact by using a *force sensitive handle*. The user doesn't need extensive training and doesn't have to learn voice or gesture commands. The close coupling of user and robot in the control task allows the user to perceive the robot's actions while allowing him to contribute his own intentions simultaneously. The user is able to apply experiences from a common, manual trolley to the robot trolley but is supported by several additional functionalities like motor power, obstacle avoidance or local maneuvers controlled by *Force Commands*.

The *force sensitive handle* (see Figure B.16) is part of the robot's multimodal user interface. It is used to manually steer the robot while operating in the *Manual Steering Mode* as well as to give *Force Commands* to the robot which are then executed autonomously.

### B.6.1. Sensor concept

The basic concept of the *force sensitive handle* is to design a system to detect the applied forces and torques, gather them and transfer them to a classification mechanism. Following the classification, motion commands are generated and transferred to the behavior network which is responsible for the motion generation.

To sense and detect the user's intention via the trolley's handle a physical contact of user and trolley is necessary. The user grabs the handle at least with one hand. The possible area for the physical contact of the user is illustrated in Figure B.17 (left) by the yellow rectangle. The most typical areas for the contact are

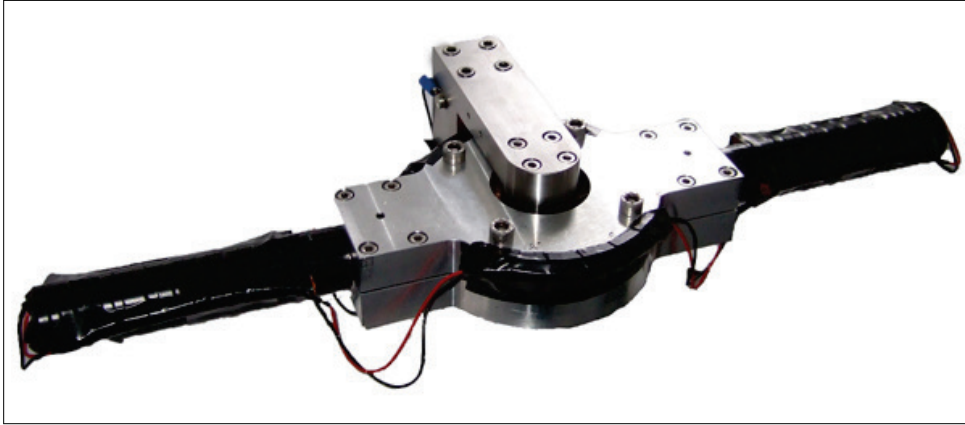


Fig. B.16.: The *force sensitive handle* (third version) for manually steering the robot

illustrated by the orange circles. The arrows demonstrate the most typical direction of the forces applied by the user.

The user is able to apply forces as well as torques on the trolley’s handle using either one or both hands. During contact of the hands, the rigid body of the handle combines the individual forces and torques to a single 3D force vector (see Fig. B.17 (middle)). For steering the robot, but also for the classification of the intention, only the projection of the force on the horizontal plan (Fig. B.17 (right)) as well as the torque around an axis parallel to the normal vector of the horizontal plan is used.

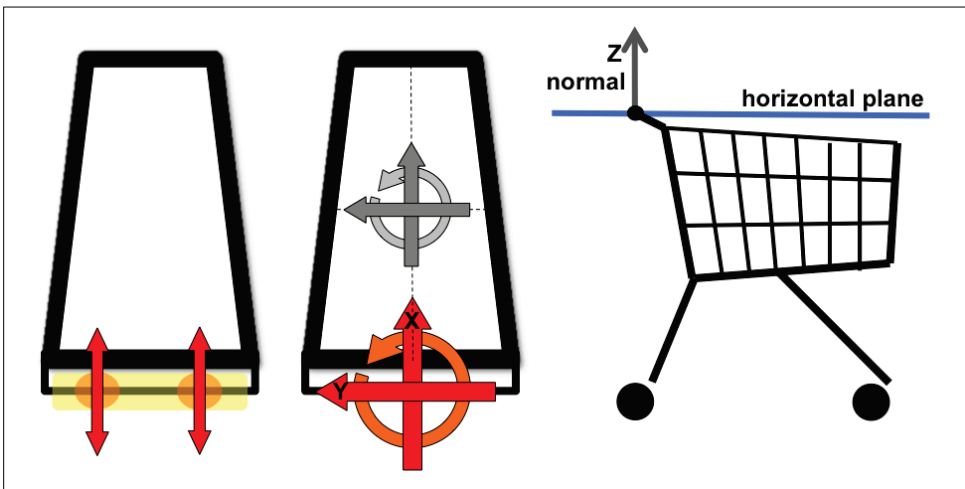


Fig. B.17.: Forces relevant for the *Manual Steering Mode*. Left: Forces typically applied by human users on the handlebar of a shopping cart. Middle: Forces (red arrows) and torque (orange circle) that are finally provided by the handle. For steering the robot these have to be transformed into the robot coordinate system (grey) – a Y-force on the handle results in a torque on the shopping cart. Right: Horizontal plane and normal vector for the decomposition of forces and torques.

In detail the process is as follows (see Fig.B.18): The forces that the user exerts on the bars of the handle are transferred into the sensor. Here they again exert a strain on strain gauges (strain sensitive resistors) mounted on thin metal beams. The strain is measured by ICs<sup>2</sup> connected to the strain gauges. The values

<sup>2</sup>Picostrain PS021, Acam-Messelektronik GmbH

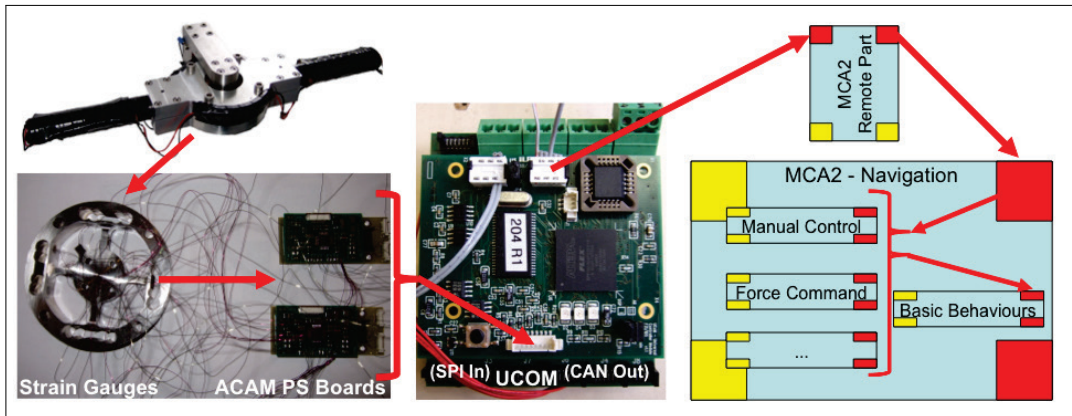


Fig. B.18.: Processing tool chain of the *Manual Steering Mode*: handle with strain gauges - ACAM-Board - UCOM-Board - MCA2 Remote part - Module Manual Control - Module Force Command Classification - Advanced Behaviors - Basic Behaviors

are transferred to a UCOM-board [130] via SPI. Here the several measured forces are transformed into two forces and one torque, which is the most significant information used in the *Manual Steering Mode*. Additional signals are provided by two buttons (or arrays of micro switches) on the handle that are used as dead man's switches. The UCOM-board transfers the resulting forces to the navigation system if at least one of the buttons is pressed, only. The second functionality of the buttons is to trigger the classification of *Force Commands*. At the moment when both buttons are released the responsible classification mechanism processes the forces measured in the last second and classifies the given command. While at least one of the buttons is pressed the actual command is classified as *manual steering* and the forces are transformed into movement commands. Details on the data processing in the *Behavior-Based Control* have been described in Chapter 6.4.1.

### B.6.2. Three iterations of the *force sensitive handle*

Three iterations of the handle have been developed until reaching the final design (see Figure B.19 for an overview). The first version was intended to be quickly available as it was needed as basis for the development of the control software. Thus, the design has been simplistic. The second version was designed to perform first experiments and thus had to be more robust and precise. It was outfitted with two H-shaped force sensors instead of the single beams of the first version. This version tended towards having internal stress on temperature changes and lost its calibration too frequently. On the gathered experiences the last and final version has been designed. This time, a single circular force receptor made out of one piece was used to prevent inner stress. Several finite element simulations have been run to get the best shape for the sensor and the best position for mounting the strain gauges. Intending to make the third version the final one, robustness in processing the acquired data was put into the focus. The number of strain gauges as well as the number of measurement boards was doubled. Additionally, shielded cables and electronic boxes have been used to ensure the quality of the very sensitive signals. The thorough analysis of the issues of the second version of the handle and the design of the third version is described extensively in the internship report of D. Böttinger [Boe10], the the calibration process of the handle in the internship report [Rit10b], and the design of the first handle in [Ost10].

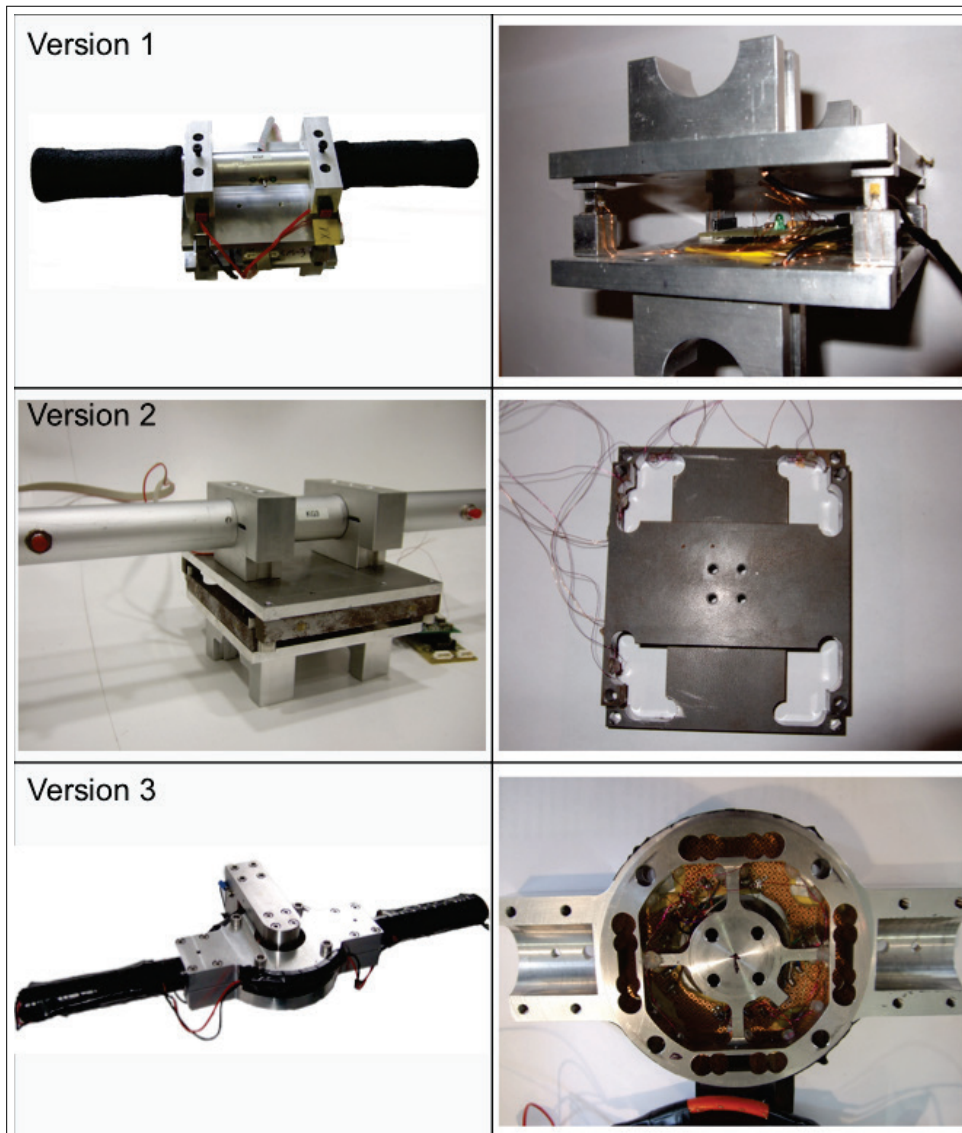


Fig. B.19.: The three versions of the *force sensitive handle*. Left: the complete handle. Right: closeup of the sensor. V1: early prototype for a quick start with the development of software components. V2: first experimental version of the handle. V3: final improved version of the handle using a double amount of strain gauges, an inlaying board and arrays of micro switches to substitute the dead man's switches.

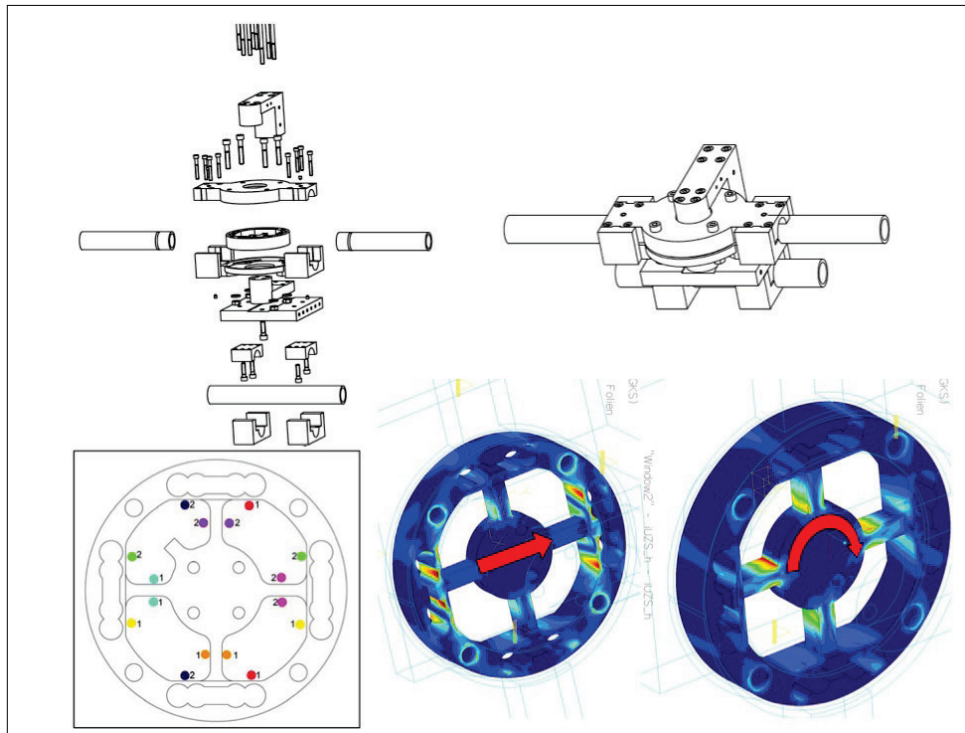


Fig. B.20.: Construction of the third version of the *force sensitive handle*. Top: mechanical construction sketch. Bottom left: placement of the strain gauges. Bottom right: finite element simulation showing the strain on the sensor when applying forces and torques as indicated by the red arrows.

**Development of the final version of the *force sensitive handle*** While conducting tests, the second version of the *force sensitive handle* showed four kinds of issues which had to be tackled when designing the third version (see Fig. B.19 and B.20):

- **Decomposition of applied forces:** The handlebar was located above the sensor, thus, there was always a strong torque, even when the user tried to exert forces in the X-Y-plane only. In the third version, the sensor has been mounted in the center of the handlebar.
- **Torque:** The torque around the Z-axis was difficult to measure due to the positioning of the strain gauges. In the third version, they have been applied on radial beams as well.
- **Inner stress:** The sensor was not made out of one piece but first mounted by screws and later by bolts of the same material as the sensor. Nevertheless, changes in the environmental temperature resulted in significant stress in the sensor which was measured by the strain gauges. This forced a re-calibration of the sensor after significant temperature changes – in the summertime often two times a day. In the third version, the sensor has been made out of one piece.
- **Sensitivity of analogue signals:** The strain gauges are actually strain-sensitive resistors. The difference in the specific resistance depending on the strain is rather small. Thus, noise generated by external influences can overpower the actual signal. In some cases the impact of just holding the hand closely to the wires caused a signal of five times the strength of the force applied. In the third version, the first board has been integrated into the housing of the sensor and shielded cables have been used. Additionally, the number of strain gauges has been doubled and an additional strain gauge has been installed to measure the strain caused by the temperature changes themselves.



## C. Complementary System Components

This chapter gives an overview on complementary system components which have been developed in the course of this thesis in order to have – seen from an application-oriented point of view – a complete system at hand. These components shall facilitate testing the core of the control system in the context of the scenario of application in general and conducting user studies in particular. Furthermore, for some components additional third-party alternatives have been integrated, preventing overspecialization of the control system and demonstrating the system’s modularity and extensibility.

The components summarized here are located in the application-specific part of the control hierarchy or used for data acquisition. They are considered being auxiliary components, only, and will have to be substituted with more specialized and sophisticated ones, once the system should actually be applied.

**Organization of this chapter:** To successfully perform complex assistance tasks such as following and guiding in the store, the robot must be able determine the current position of its user. Furthermore, other moving objects have to be taken into account, too. The first three section will focus on systems for tracking objects or persons: Section C.1: “Object tracking based on the occupancy grid and planar laser scanners”, C.2: “Detecting and tracking the user using onboard sensors”, and C.3: “The *intelligent environment*” which tracks objects with cameras and/or laser scanners distributed in the environment.

The second part of this chapter will focus on means to control the *service robot* namely the user interfaces and the application-specific *communication layer*: Section C.4: “The *InBOT-UI*”, C.5: “The *CR-UI*”, and C.6: “The *TaPAC Client*”.

The chapter will be concluded by introducing a graph-based navigation system in Section C.7.

### C.1. Object tracking based on the occupancy grid and planar laser scanners

For a *service robot* operating in a dynamic environment it is crucial to be able to identify moving objects and to track their motions. The first approach uses the local occupancy map of the robot’s *local world model* which is built based on the sensor reading of the two planar on-board laser scanners<sup>1</sup>. In contrast to the *intelligent environment* introduced in the next section, using only on-board sensors has the advantage of having no installation effort, especially in large-scale environments. But it comes at the costs of sensor readings with less quality and a disadvantageous perspective, resulting in frequent occlusions.

In the following, the algorithm for detecting, matching, and tracking moving objects is introduced. This algorithm is based on the coordinates of the objects’ positions extracted from the occupancy grid. First the grid maps are pre-processed. Then the map is segmented to distinguish objects from free space. Finally, the center of gravity (COG) of the objects is determined and matched with a predicted COG, generated from the previous frames. The successfully matched objects are updated in the *local world model*.

---

<sup>1</sup>Two SICK S300 or LMS100, respectively, each with a planar 270° field of view

### C.1.1. Pre-processing of the grid map

First of all, a copy of the *local world model*'s local occupancy grid map is acquired. By first dilating the map – which is interpreted as binary image – all the objects whose distance is smaller than two cells are merged.

Afterwards, by eroding the map small objects only consisting of few cells like legs of a table or a chair are deleted from the map because they are most likely not moving. Taking them into account would mean having to match vast amounts of tiny obstacle which actually might even seem to be moving due to the fuzziness of the robot's self-localization and the discretization of the environment into cells. Therefore, all objects are eroded and all objects which have less than three cells are deleted. Depending on the actual scenario of application, particularly this step has to be re-factored, depending on the expected size and type of obstacles.

Finally, the grid is segmented and all objects are extracted. For each object the Center of Gravity (CoG) is calculated using the center of the bounding box of all occupied grid cells.

### C.1.2. Tracking objects

To calculate the motion of the objects the association of the individual objects is determined by comparing the objects with predictions corresponding to the objects of the preceding frame. The objects are tracked and finally velocity and heading of the objects is computed using their displacement over several frames.

Some objects partly overlap regarding their position in the current and the preceding frame. Actually, this is highly probable as the algorithm has a cycle time of three runs per second. Therefore, an object with a size of 30cm traveling slower than 1m/s will probably overlap in two succeeding frames. Objects with an overlap of at least 90% or an average displacement of the COG of less than 10cm are considered as (almost) static objects and, thus, discarded.

If there is a unambiguous overlapping between two frames, these objects are matched. For the remaining objects a prediction based on their past motion is calculated and a gate is defined to reduce the number of possible matching candidates. From all the candidates the best fitting one is chosen, using the actual distance between COG and the predicted position as well as the size of the bounding box as measure.

In a post-processing step objects are split or merged, if necessary. Furthermore, an inertia value is calculated for each object. Because of inaccuracy and discretization, a moving object might look like a static object in several frames and afterwards move again. On the other hand, a static object could also look like a moving object. When an object has performed a significant motion, its inertia is raised, else it is decreased. As long as the inertia of an object is above a threshold, the object is considered as moving object. If the inertia of an object is zero or it could not be matched anymore it becomes a virtual object. The virtual object continues its motion with constant velocity and heading for few seconds, continuing its tracking history. This way, temporarily occluded objects will not be lost – rather, they remain in the robot's memory. This enables the robot to avoid collisions with moving objects which are occluded for some seconds.

### C.1.3. Updating the tracking history

The tracking history contains the last positions of an object and is stored in the *local world model* to be used by the individual behaviors. After going through the described process, the history is extended either by the new position of the successfully matched object or by continuing the movement of a virtual object. If



an object stays 'virtual' for a defined amount of time it is discarded completely and the tracking history is deleted. Two examples are depicted in Figure C.1:

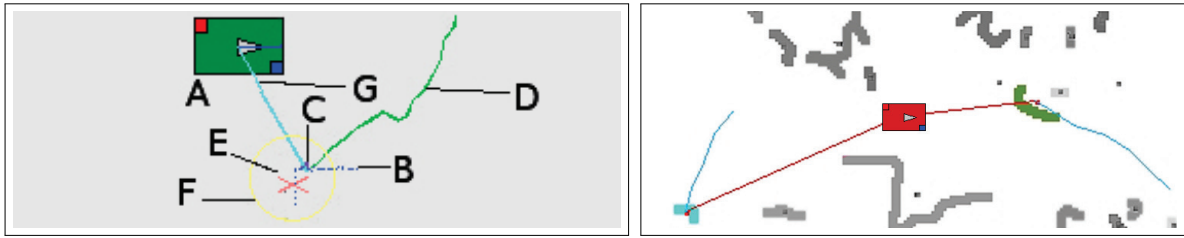


Fig. C.1.: Left: Example of a tracked object, accompanied by a prediction for the succeeding frame. A: robot *InBOT*, B: laser scanner readings of an object (current frame), C: the object's gravity center (preceding frame), D: the object's track-line, E: the object's predicted position (shape/size of X: uncertainty), and F: the circular gate around the new COG of an object in the succeeding frame. All prediction points (center of X) in the circle will be considered as potential position in the succeeding frame.

Right: Occupancy map with two moving objects (colored) moving around the robot in our labs. The lab is filled with lots of chairs, tables, PCs, bags, etc. Several obstacles are not dense, so the laser range finder is able to see through them.

## C.2. Detecting and tracking the user using onboard sensors

When performing assistance tasks such as following and guiding in the store, the current position of its user is of crucial importance. Accordingly, the robot's user is a moving object of special interest, thus, identifying the position of moving objects is not sufficient, the identification of the user is necessary.

In this section, a system to track the robot's user is introduced, which has been developed by T. Germa at LAAS CNRS and has been integrated on *InBOT*. In [59] more details are provided for the problems of detection and tracking the user using on-board sensors, more precisely a mono-camera and a long-range RFID system.

The robot is equipped with a long-range RFID reader connected to eight directional antennas and with a camera mounted overhead on a pan/tilt unit (PTU) to track the user. The user carries an RFID transponder, the user identification is performed based on the transponder ID. This makes the identification of the user more robust, especially when the user has been occluded for some time. For the identification of the user based on visual data, a user appearance model using color templates of the face and the clothes is initially learned (see Fig. C.2). The image is searched for regions of the learned color and a particle filter is used to find the best fitting position of the user in the image. Figure C.4 shows the individual components used for the calculation.

The relative human/robot pose can be determined by tracking the detected user. First, the user's RFID transponder is detected. Figure C.3 presents the calibrated reception fields: the antennas receive signals in a 120deg cone with an effective range of 4.5 meters. A tag can be identified to be inside the red, blue, or green regions according to which antennas actually received the signal. This gives a rough estimation of the users current position by which the PTU can be adjusted if the user is not currently tracked. In parallel, the visual data is analyzed. The relative angular position is given by the current orientation of the PTU and the camera model. The relative distance is coarsely estimated from the face width and height in the image

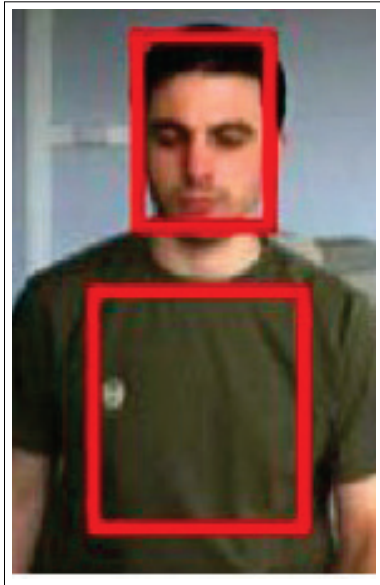


Fig. C.2.: The color template used to memorize the color and size of the user's face and clothing (image source: [63]).

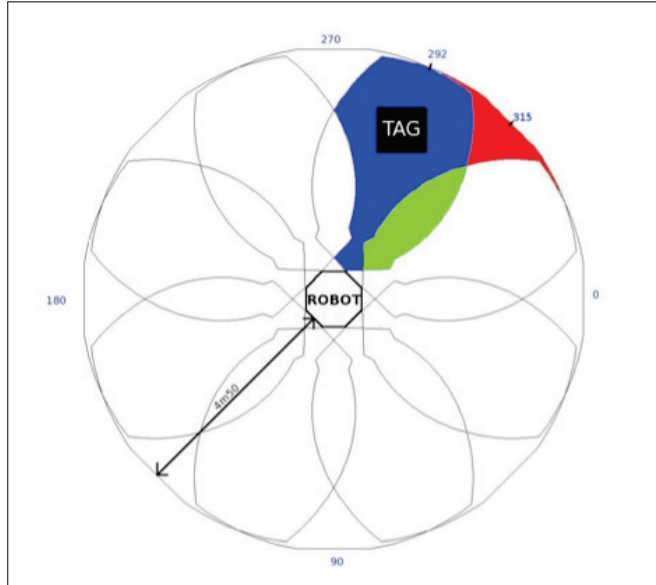


Fig. C.3.: Arrangement of the eight directional RFID antennas on the robot (image source: [59]). The RFID tag carried by the user can be located to be either inside the blue, green, or red area.

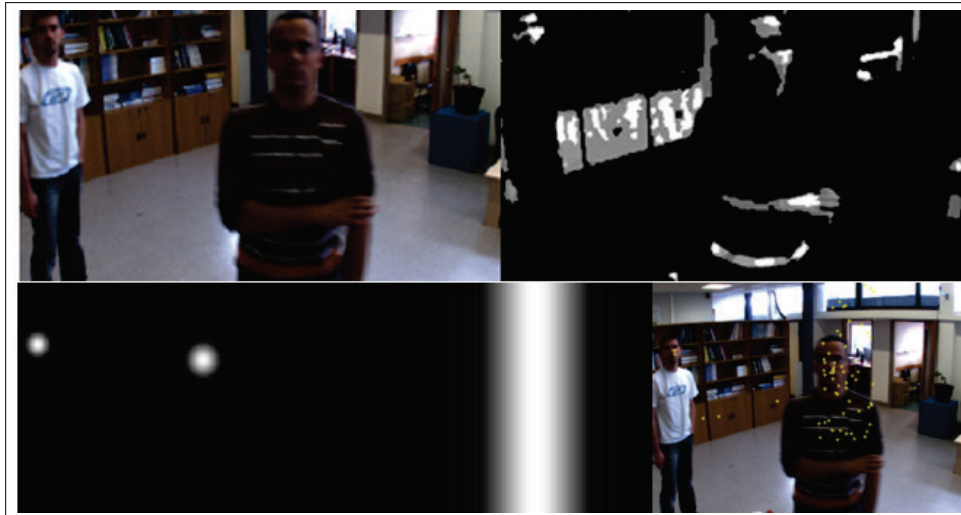


Fig. C.4.: User tracking by on-board camera (image source: [59]). From left to right: original image, skin probability image, face detection, azimuthal angle from RFID detections, accepted particles (yellow dots) after rejection sampling.

compared to the one learned a priori. Finally, Figure C.5 shows the robot *InBOT* following a user by the described means.



Fig. C.5.: *InBOT* is following a user who is carrying an RFID tag in his hands by means of RFID localization and color templates of skin and clothes. The white boxes mounted on the basket house the RFID antennas.

### C.3. The intelligent environment

In contrast to the methods using on-board sensors only, this section focuses on sensors distributed in the environment. The advantage is that the sensors can be placed with favorable fields of view (e.g. making occlusions less probable), the sensors do not move themselves resulting in sensor readings of better quality and enabling background subtraction techniques, and finally the complete environment can be monitored, not only the close vicinity of the robot or robots, respectively.

There are two main components which contribute to the system (see Figure C.6). The first one are laser scanners mounted on the wall at chest-height. The tracking algorithm is the same as described in Section C.1 “Object tracking based on the occupancy grid and planar laser scanners”, but has been extended with a background subtraction algorithm. And the second component is a vision-based approach using cameras mounted at the ceiling. These have a even better field of view and could additionally be used to classify the tracked objects or to identify the robots’ users by applying classifiers based on the person’s clothes or face. For the FZI labs two laser scanners<sup>2</sup> and four cameras<sup>3</sup> have been installed along with a PC responsible

<sup>2</sup>SICK LMS100, each with a planar 270° field of view

<sup>3</sup>Vivotek IP8151

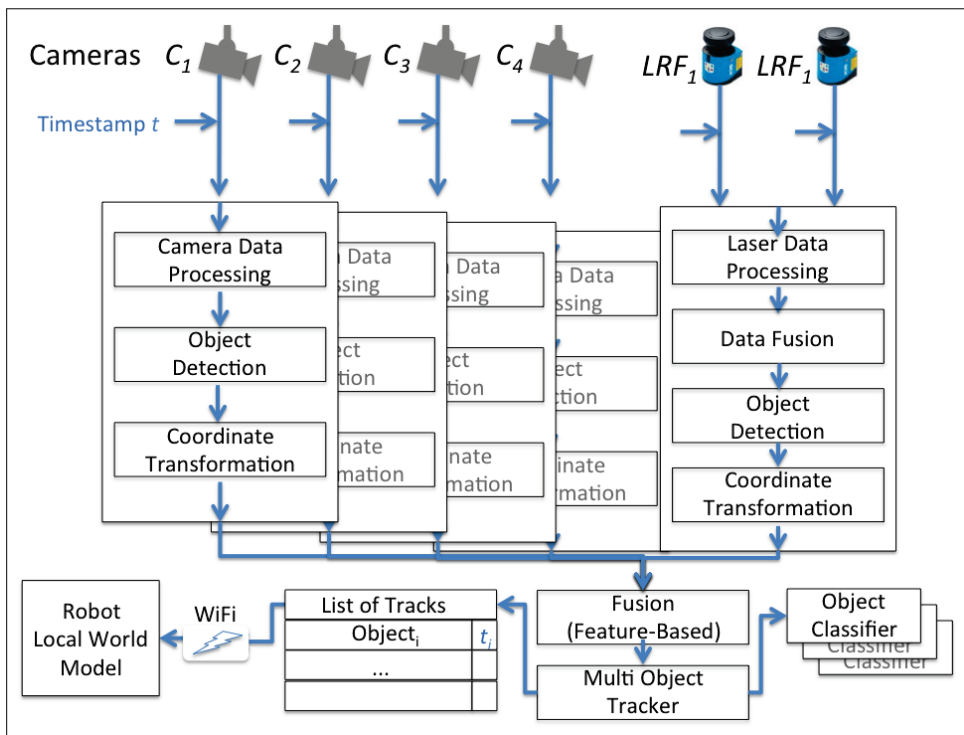


Fig. C.6.: Architecture of the vision- and laser scanner-based system for tracking moving objects using sensors installed in the environment.

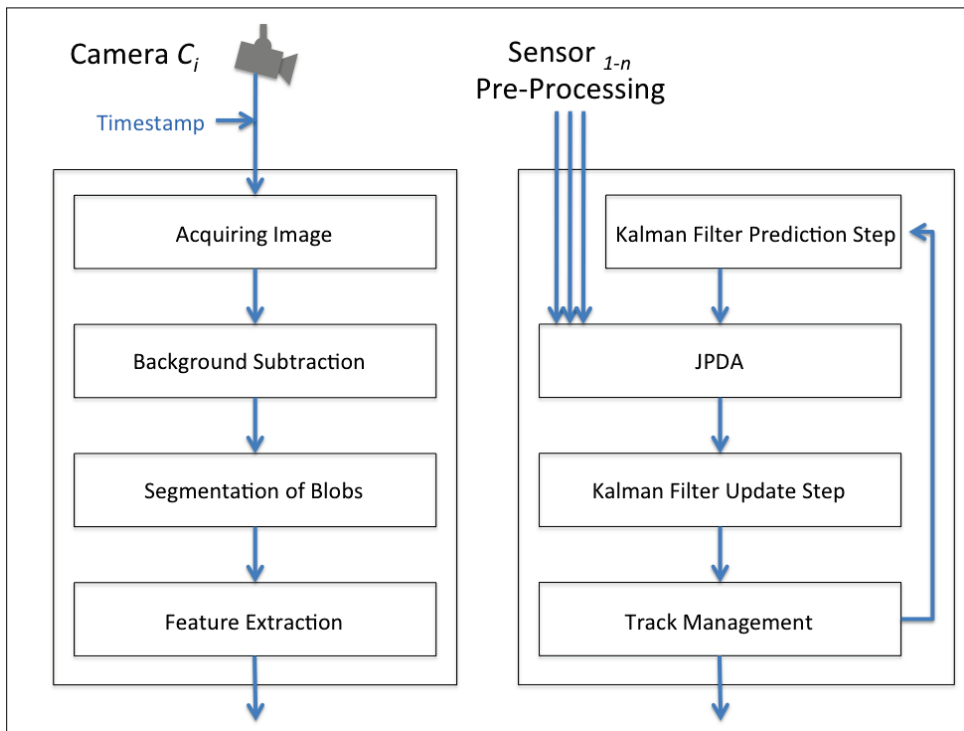


Fig. C.7.: Data processing pipeline for the vision-based part of the person tracking: background subtraction, generation and processing of blobs, feature extraction, kalman filter prediction step, data association, kalman filter update step, and management of tracks.

for the data processing and the communication with the robots. Extensive information can be found in the diploma thesis of F. Stehle [Ste11].

The processing pipeline for camera data and person tracking is depicted in Figure C.7 along with examples for some steps in Figure C.8.

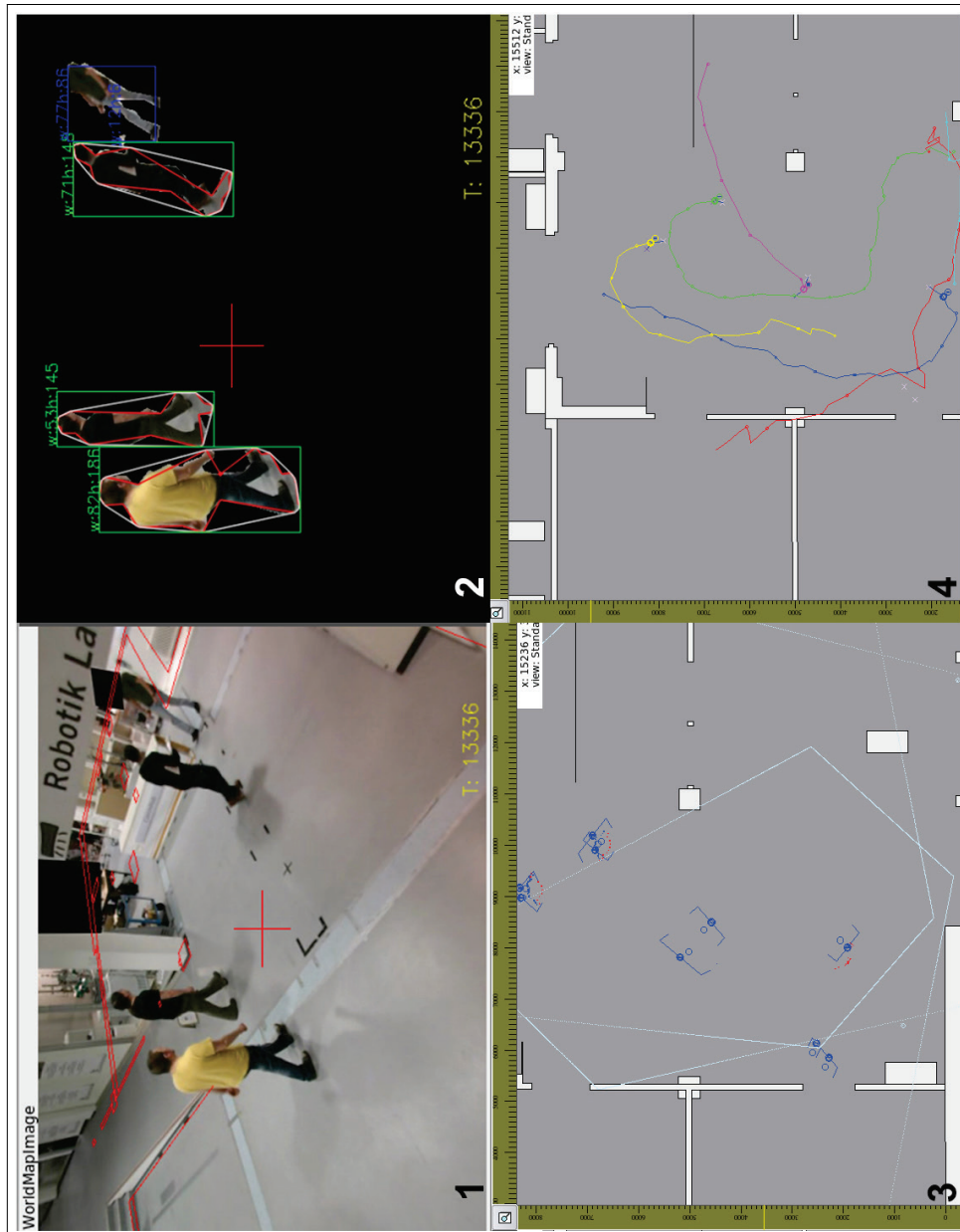


Fig. C.8.: Examples of the individual steps of the person tracking. The same scene shown in four different steps: (1) image from one of the cameras, (2) foreground blobs with bounding boxes and convexity defect check, (3) MCAGUI view (top view) of 5 persons, each seen from several sensors (blue lines: bounding boxes seen from top, dots: laser readings), and (4) track lines for 5 persons.

**Processing pipeline for camera data:** As the environmental sensors are fixed, first of all the a background subtraction is performed to identify foreground objects in the image. Here the *codebook* algorithm

[91] is used, which is robust versus shadows cast by objects. Initially, it memorizes the scene which should contain background objects, only. Afterwards, an online learning component slowly adapts the memorized scene to changes.

In the next step continuous blobs are segmented. Clusters of foreground pixels are merged by morphological operations (open, close) and then connected to continuous *blobs* by *connected component labeling* [74]. These *blobs* are evaluated regarding their size and estimated distance to the camera – assumed that all objects are actually touching the floor. This way false positives can be discarded. Finally, the convexity of the objects is analyzed [72] like done when analyzing the pose of a human hand [106]. *Blobs* which are too wide and have two strong convexity defects regarding the vertical axis which are opposing each other are considered to be actually two persons and, thus, split into two objects (Fig. C.9 (left)).

The final step of the vision data processing is the extraction of various features which can later be used for the data association like height and width of the object but also its color, or more precisely its color pattern.

**Tracking multiple objects** For each individual tracked object one *kalman filter* is kept which is continuously updated in a cyclic manner. The first step in tracking the objects for each cycle is to perform the prediction step for each of the filters, assuming a linear motion model of the objects.

In the second step a *joint probabilistic data association filter (JPDA)* is used to correlate the detected objects of the individual sensors with the predictions of the already tracked objects. A gate of  $2m$  is used to reduce the amount of possible candidates. Based on a candidate correspondence matrix the  $n$  best hypotheses for sets of matches are calculated. The probability for each pair to be a match is calculated by summing up

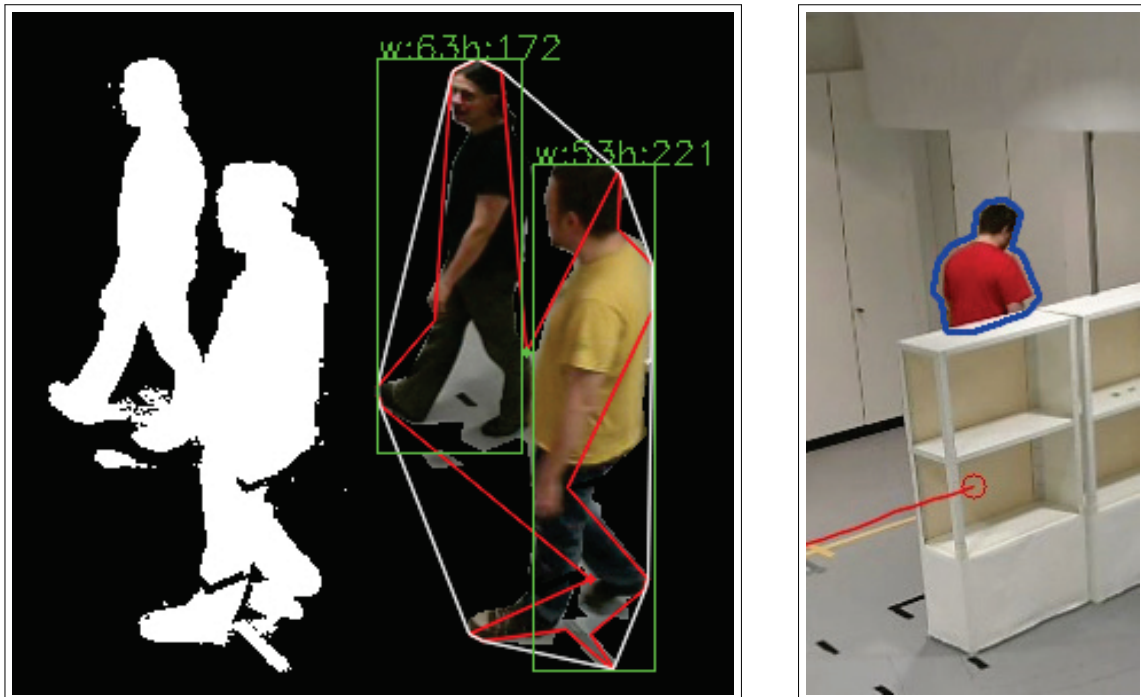


Fig. C.9.: Special measure are taken to handle specific situations.

Left: By identifying convexity defects regarding the vertical axis (red versus white line), objects which have a merged *blob* (left) can be separated (green bounding boxes).

Right: By assuming that every object touches the floor and by merging the information from several sensors, even partially occluded objects can be tracked with a correct X-Y-position.

the number and confidence of the occurrence of this match in the individual hypotheses and by favoring the best matches to avoid the track coalescence problem of objects with parallel paths. Merging several sensors facilitates tracking even partially occluded objects with correct coordinates (Fig. C.9 (right)).

Afterwards, the resulting set of matches is used for the update step of the *kalman filters*.

Finally, the track management takes place. Here new tracks are initialized when a object could not be matched, old tracks are deleted, or virtual objects are created and tracked for a defined amount of time when the objects could not get a new match because they are probably occluded.

#### C.4. The *InBOT-UI*

A central component for human robot interaction is the user interface (UI). Especially for managing complex content like shopping lists or a recipe database, a graphical interface is crucial. The UI designed for the robot *InBOT* (see Figure C.10) implements the architecture's application-specific part – the *communication layer* – and provides for the application logic. It is meant to be run on a touchscreen PC to ease its use. Additionally, a bar code scanner can be used to get information on products and a speech output informs the user on special events. Extensive information can be found in the mid-study thesis of A. Gorbunov [Gor12]



Fig. C.10.: A customer using the *InBOT-UI*

The work flow for using the GUI is designed to be as follows (but the user can divert if he wishes to): The user logs into the system by scanning his ID card and entering a PIN (Fig. C.11 (left)). He is then asked if he wishes to load an old shopping list or to start a new one. Afterwards he can enter new products. When entering a product, a list of choices for the product is presented (e.g. different kinds of milk available) and the user can enter the number of pieces he wants to shop. While entering products he is advised to what other customers have bought with a similar choice of products. Additionally, the user can access a database of recipes, either by browsing the list or by asking the system which recipe fits best to the products currently

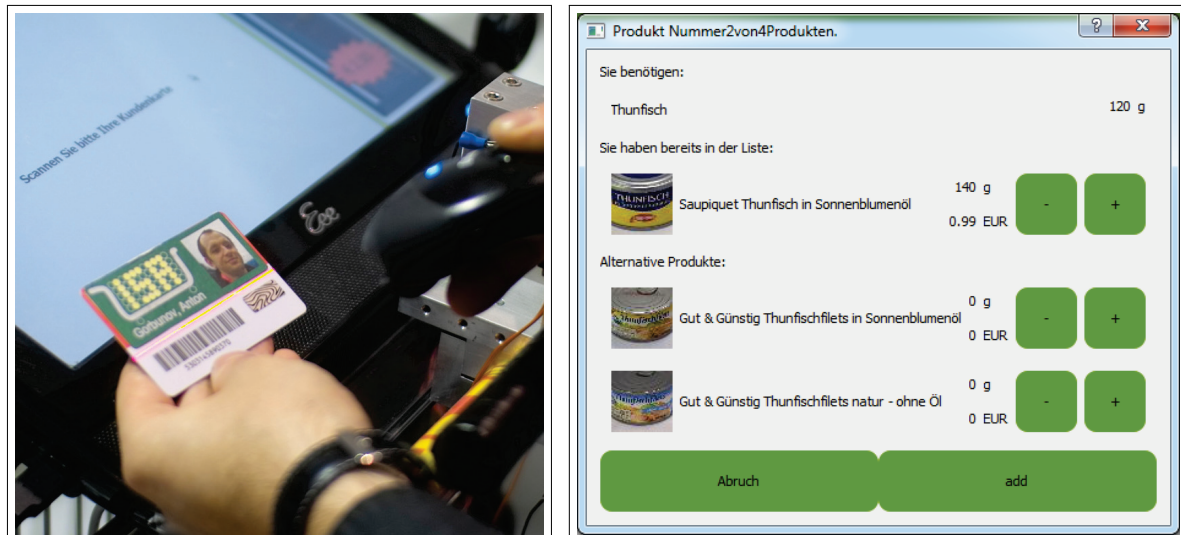


Fig. C.11.: Left: the user logs into the system by scanning his ID card and then entering a PIN. Right: dialogue informing the user that tuna is needed for the selected recipe, that some kind of tuna is already in the shopping list and asking if another piece of tuna shall be added along with the choices of tuna available in the shop.

on the shopping list. After selecting a recipe and entering the number of persons for the meal, further goods are added to the shopping list (Fig. C.11 (right)). Finally, the system presents the main navigation screen (Fig. C.12) and guides the user to the products. When reaching a product a speech output is uttered and a pop up window informs the user on the screen. Here the user can increase or reduce the number of pieces he puts into the basket. The product is added to the “done” list when either the user presses the done button in the dialogue, scans the product or after a couple of seconds when neither the bar code scanner nor the touch screen are used. When scanning any product in the meantime, the GUI displays information on the product. After finishing the shopping trip, the user commands the robot to queue up at the checkout counter.

Figure C.13 shows the architecture of the interface system. All top-level screens are available from the main screen and the user is guided through the sub-screens by defined work flows. Based on sparse events, the system can jump to specific sub-screens directly, for example when a product’s bar code has been scanned. Additional events are received via the *command interface* (see Chapter 3.3.3) which connects the *InBOT-UI* – which is actually a implementation of the *communication layer* – with the main control system. The speech output is implemented by the open source package *Espeak*<sup>4</sup> and informs the user about events received from the main control system.

The *InBOT-UI* uses a database which contains all products along with their location in the shop as well as a *topologic-metrical map* of the shop. When the user finishes editing the shopping list, a graph is generated. This graph contains the center of all topological areas, a navigation point in front of the links between the areas and the products in the shopping list. Starting from the robot’s current location, a greedy algorithm is then used to arrange the selected products based on their distance along the graph, favoring products located nearby and on the same side of the alley. When the user starts the navigation, the corresponding change of the *mode of operation* and the target location is handed down to the control system using the *command interface*.

<sup>4</sup><http://espeak.sourceforge.net/>



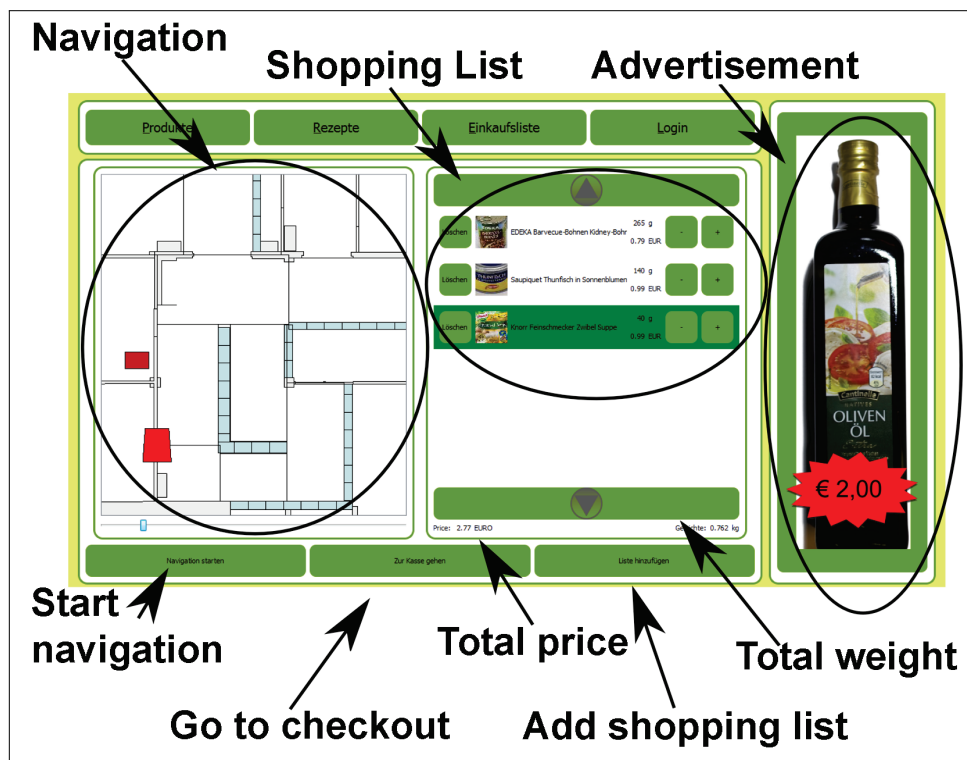


Fig. C.12.: The main navigation screen showing a zoom-able map, the shopping list, and a set of controls.

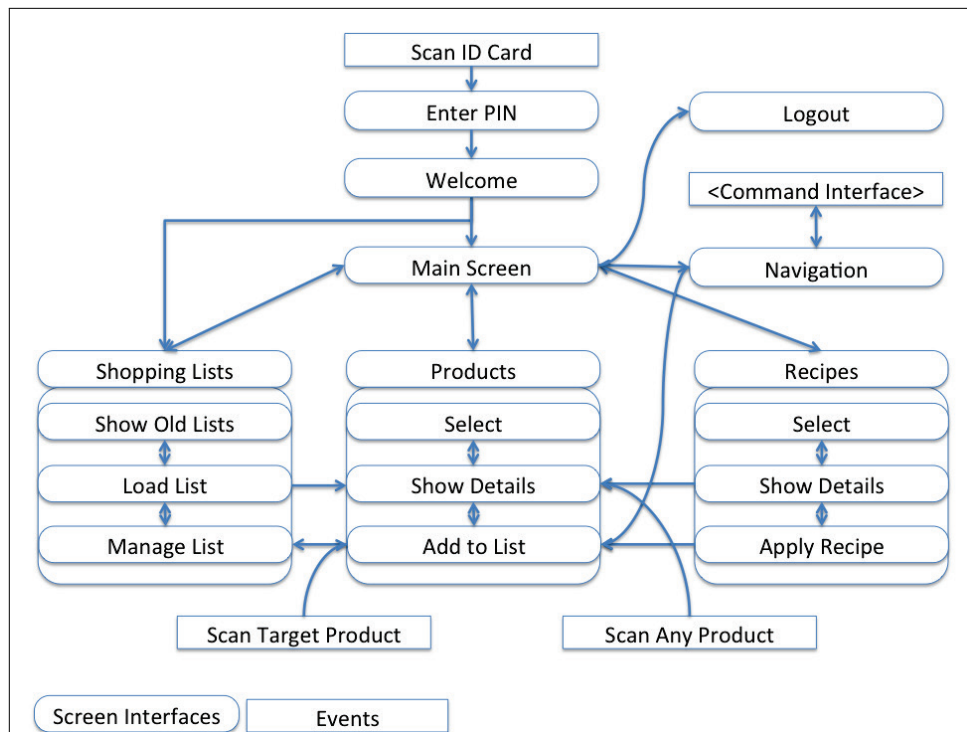


Fig. C.13.: Architecture of the *InBOT-UI*.

### C.5. The CR-UI

An alternative user interface (UI) is the multimodal *CR-UI* (Fig. C.14). It has been integrated on *InBOT* implementing the *communication layer* while the robot was placed at the disposal of the project “Comm-Rob” until the CommRob demonstrator was available. The interface was developed by Prof. H. Kaindel’s group at TU Vienna based on Java and is attached to *InBOT*’s control system via the *command interface* (see Chapter 3.3.3). What distinguishes this user interface is the claim that the content is generated (semi-) automatically based on UI and discourse models defined in UML [41] [87]. The UI accepts several methods for interaction: a touch screen interface, bar code scanner, speech output, and even speech input using the grammar-based speech recognizer *Julius*<sup>5</sup>.

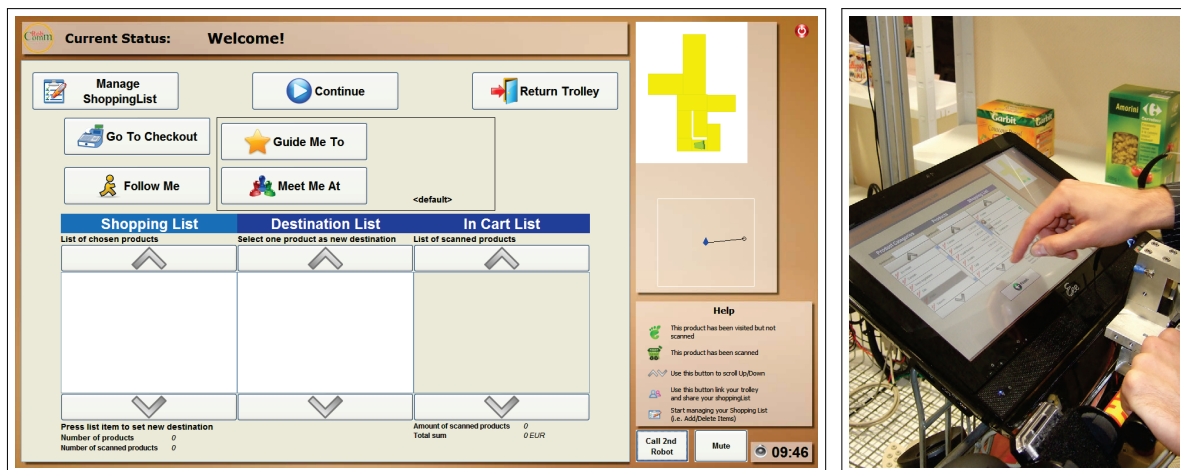


Fig. C.14.: Kaindl’s *CR-UI* on *InBOT*. Left: Main navigation screen with the lists for products to be bought, which have not been taken yet, and which have already been taken. Coming along with the list is the map in the top-right corner – here of the FZI labs – showing the current location and the location of the target product. Right: Foto showing a customer using the UI on *InBOT* in a small simulated shop set up here in the FZI labs.

The work flow is as follows: The user logs into the system by pressing the corresponding button on the touch screen interface. Then he enters his shopping list in the order in which he wants to visit the products. After finishing the shopping list he can give the command to be guided to the first product, either by pressing the corresponding button or by saying “Guide me to the next product”. He can also select a specific product instead of the next one. When reaching the product the user is informed by a speech output. The product is marked as “in the basket” when it is scanned with the bar code scanner. The user can increase the number of pieces by scanning it several times. When user an robot pass a product which is on the shopping list while going to another product the UI commands the control system to slow down and informs the user via speech output ([20] and [78] show some results form an human factors’ perspective). After finishing the shopping trip, the user commands the robot to queue up at the checkout counter. The UI then starts a dialogue with the checkout system and transfers the information on the shopped products.

<sup>5</sup>[http://julius.sourceforge.jp/en\\_index.php](http://julius.sourceforge.jp/en_index.php)

## C.6. The *TaPAC Client*

The *TaPAC* (*Transports and Peripherals Administrative Centre*) [144] has been developed at FZI (IDS) to manage transportation tasks for fleets of *automated guided vehicles* (AGV). Regarding the architecture presented in this thesis, the *TaPAC* client represents the application specific component – the *communication layer* and particularly the application logic. Usually it is used in combination with a HTFM graph-based navigation which is introduced in the succeeding section. The *TaPAC* client has been integrated with the control system of this thesis to be applied on the transportation robot *Odete*. The client can be commanded directly using a network-based GUI or – as done in larger installations – via the *TaPAC* server which manages the fleet of AGV and controls automated doors and lifts.

## C.7. The IDS graph-based navigation system

The graph-based navigation system (Fig. C.15) has been developed at FZI (department IDS) to enable *automated guided vehicles* (AGV) to operate without physical additives such as optical or magnetic guidance lines [144]. The robots traverses the graph node by node, directed by a server which is responsible for the fleet organization, if necessary. The graph is for example used to implement two-way traffic in corridors or to define the robots' paths in larger halls. The edges of the graph can be annotated with additional information like the maximum allowed velocity. The graph itself is implemented by a so-called *HTFM* (*Hierarchical Topological Finite State Machine*). Regarding the architecture presented in this thesis, this navigation system represents the *strategic layer*. In combination with the *TaPAC Client* it has been integrated with the *tactical* and *reactive layer* of this thesis on the transportation robot *Odete*.

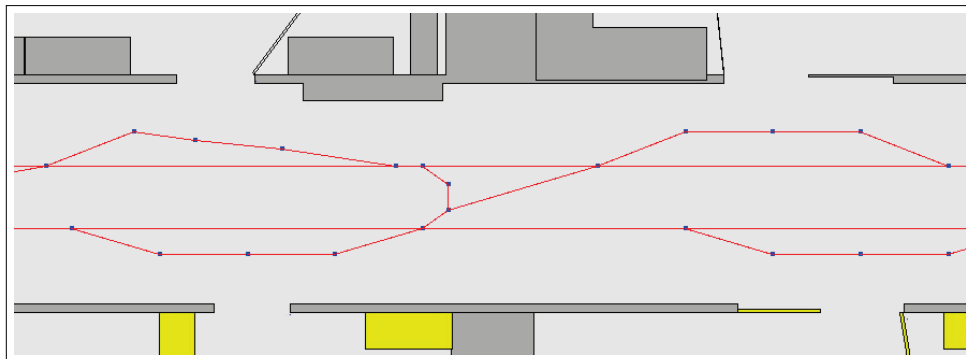


Fig. C.15.: The IDS graph-based Navigation system for AGVs.



## List of Figures

1.1	Movie Metropolis . . . . .	1	2.23	Care-O-Bot . . . . .	50
1.2	Industrial robot Unimate . . . . .	1	2.24	enon . . . . .	50
1.3	<i>TransCar</i> automatic guided vehicle (AGV) by Swisslog . . . . .	2	2.25	Real MEA App . . . . .	51
1.4	Automated subway train by Siemens . . . . .	2	2.26	EDEKA App . . . . .	51
1.5	Window cleaning robot RACOON . . . . .	2	2.27	Giving Cart . . . . .	51
1.6	Rhino and MINERVA . . . . .	3	2.28	Concierge for Cart . . . . .	52
1.7	Robox . . . . .	3	2.29	U-Scan Shopper . . . . .	52
1.8	TOOMAS . . . . .	3	2.30	Smart Cart . . . . .	52
1.9	Supermarket with non-static objects . . . . .	6	2.31	KleverKart . . . . .	52
1.10	Supermarket: removable special offer . . . . .	6	2.32	IBM Personal Store Assistant . . . . .	52
1.11	InBOT and ETrolley . . . . .	7	2.33	ETrolley . . . . .	52
1.12	Sketch: Holonomic drive parking . . . . .	9	2.34	TOOMAS . . . . .	55
1.13	Sketch: Holonomic drive dodging . . . . .	9	2.35	RTS . . . . .	55
1.14	Design concepts . . . . .	11	3.1	Pedestrian Behavior Hierarchy . . . . .	59
1.15	Interplay of main components . . . . .	16	3.2	Architecture from Pedestrian Model . . . . .	63
1.16	The robot <i>InBOT</i> . . . . .	16	3.3	Orthogonal control flow . . . . .	64
2.1	Pedestrian Behavior Hierarchy . . . . .	22	3.4	<i>control sharing</i> and <i>modes of operation</i> . . . . .	64
2.2	Introduction of edge-meshes as motion model of pedestrians . . . . .	23	3.5	The robot's control architecture . . . . .	65
2.3	Example: Fundamental Diagram . . . . .	24	3.6	The infrastructure . . . . .	66
2.4	Simulation of traffic jams . . . . .	25	3.7	<i>local world model</i> . . . . .	67
2.5	Density tolerance . . . . .	25	3.8	control architecturewith interfaces and data flow . . . . .	69
2.6	<i>BBC</i> Fusion Mechanism . . . . .	30	4.1	Behavior hierarchy of the control system . . . . .	74
2.7	Behavior Module . . . . .	33	4.2	Concept of the control system . . . . .	75
2.8	Fusion Behavior . . . . .	33	4.3	Predictive obstacle avoidance . . . . .	78
2.9	Example for <i>ib2c</i> . . . . .	35	4.4	Grid map and extracted obstacles . . . . .	80
2.10	3T Architecture . . . . .	38	4.5	2D measurement wheel . . . . .	81
2.11	Architecture by Alami . . . . .	38	4.6	Sketch of RFID barriers . . . . .	81
2.12	Types of <i>topological maps</i> . . . . .	40	4.7	MCA module implementing behavior . . . . .	82
2.13	VFF, VFH, and VFH+ . . . . .	42	4.8	The behavior module . . . . .	82
2.14	Robots Hierarchy . . . . .	45	4.9	Behavior coordination . . . . .	83
2.15	Airplane cleaning robot . . . . .	45	4.10	Behavior fusion . . . . .	83
2.16	<i>MOSRO</i> inspection Robot . . . . .	45	4.11	The <i>Behavior Network</i> . . . . .	84
2.17	Cleaning robot roomba . . . . .	45	4.12	Sketch: Safety sectors . . . . .	85
2.18	The robot ARTOS . . . . .	47	4.13	Sketch: Safety calculation . . . . .	85
2.19	The robot RAVON . . . . .	47	4.14	Screenshot: Safety velocities . . . . .	87
2.20	The robot MARVIN . . . . .	47	4.15	Architecture of the safety group . . . . .	87
2.21	Rhino and MINERVA . . . . .	49	4.16	Relevance of obstacles . . . . .	89
2.22	Robox . . . . .	49	4.17	Scene using <i>Avoid Obstacle</i> behaviors . . . . .	90
			4.18	Plot of the relevant area . . . . .	90
			4.19	Matlab simulation of <i>Avoid Obstacle</i> behavior . . . . .	91
			4.20	MCA implementation of AO behaviors v1 . . . . .	92
			4.21	Comparison of <i>Avoid Obstacle</i> behaviors . . . . .	94
			4.22	<i>Avoid Obstacles</i> behavior: Example . . . . .	95
			4.23	MCA implementation of AO behaviors v2 . . . . .	95

4.24	Evaluation No1 of the reactive part of the <i>BBC</i> . . . . .	96	5.3	MCA implementation of the reactive avoidance of moving obstacles . . . . .	125
4.25	Evaluation No2 of the reactive part of the <i>BBC</i> . . . . .	97	5.4	Sketch of the distance calculation of the Evade Behavior . . . . .	126
4.26	Sketch and MCAGUI for Look for Gaps	98	5.5	Plot of the Evade Behavior . . . . .	127
4.27	MCA: Reactive task-oriented input behaviors . . . . .	101	5.6	MCAGUI showing the virtual force fields of the Evade Behavior . . . . .	127
4.28	The <i>Behavior Network</i> . . . . .	102	5.7	<i>InBOT</i> avoids shopping cart: Sketch and photo . . . . .	128
4.29	MCA: Tool chain of the <i>tactical layer</i> . .	103	5.8	<i>InBOT</i> avoids shopping cart: MCAGUI screen shots . . . . .	129
4.30	Sketch: Geometric construction for the predictive obstacle avoidance . . . . .	103	5.9	Limitations of reactive behaviors . . . . .	129
4.31	GUI: Predictive Obstacle Avoidance . . .	103	5.10	MCA implementation of the planner . . .	130
4.32	Plot of the candidates' quality . . . . .	104	5.11	Construction of the 3d map . . . . .	131
4.33	Video: Predictive Obstacle Avoidance . .	104	5.12	Path planning and optimization in 3D grid	132
4.34	Sketch of LFC subgoal generation . . . .	105	5.13	Example of the spatio-temporal planner .	133
4.35	Scene Analysis: Sketch of Goal Point Displacement . . . . .	107	5.14	Solved limitation by planner . . . . .	134
4.36	Parking positions at Goal . . . . .	108	5.15	comparison of methods for avoiding moving objects . . . . .	135
4.37	Scene Analysis: Topological navigation points . . . . .	109	5.16	Stress test regarding moving objects . . .	135
4.38	Sketch: Detection of blocked topological links . . . . .	110	5.17	Stress test to prove the deterministic manner of the behaviors . . . . .	137
4.39	GUI: Detection of blocked topological links . . . . .	111	5.18	Photos from tests . . . . .	138
4.40	GUI: Detection of blocked topological links . . . . .	111	5.19	Final test for avoiding moving objects: a shopping run . . . . .	139
4.41	Video: Detection of blocked topological links . . . . .	111	6.1	<i>InBOT</i> is guiding a user . . . . .	141
4.42	Sketch: Detection of blocked topological links . . . . .	111	6.2	<i>InBOT</i> is controlled by the <i>force sensitive handle</i> . . . . .	141
4.43	Virtual topological area . . . . .	113	6.3	Orthogonal control flow . . . . .	142
4.44	Geometric map of a shop. . . . .	114	6.4	The user interface . . . . .	146
4.45	Map divided into areas . . . . .	114	6.5	Force sensitive handle . . . . .	148
4.46	Resulting <i>topological map</i> . . . . .	114	6.6	Examples of <i>Force Commands</i> . . . . .	149
4.47	Topologic Metric Map: A <i>topological map</i> with metric and semantic annotations	115	6.7	Assistants in <i>Manual Steering Mode</i> . . .	150
4.48	Flexible plan . . . . .	115	6.8	<i>InBOT</i> follows a user . . . . .	152
4.49	Stress test of the <i>topological navigation</i> .	116	6.9	<i>InBOT</i> guides a user . . . . .	152
4.50	Evaluation of navigation system . . . . .	118	6.10	User scanning a product . . . . .	153
4.51	Facts on the second and third system test on <i>InBOT</i> . . . . .	119	6.11	Using touch screen while walking . . . . .	153
4.52	Comparison of the mesh-based model to the robot's motion . . . . .	120	6.12	GUI: User is blocking the robot . . . . .	155
4.53	control architecture implemented in MCA	121	6.13	GUI: user hinders robot . . . . .	156
5.1	Hierarchy of the modules responsible for the avoidance of moving obstacles . . . .	124	6.14	Sketch: Robot switches position with user	156
5.2	Example of tracked object . . . . .	124	6.15	Video snapshot: Robot switches position with user . . . . .	156
			6.16	Searching for user . . . . .	157
			6.17	Sketch showing control shares regarding modes and navigation system . . . . .	159
			6.18	Occupancy grid from the obstacle assistant in <i>Manual Steering Mode</i> . . . . .	160

6.19	Concept of <i>control sharing</i> vs. modes . . .	162	B.13	Evaluation of odometry . . . . .	205
6.20	Example of <i>control sharing</i> : paths . . . .	165	B.14	2D measurement wheel . . . . .	206
6.21	Example of <i>control sharing</i> : shares . . . .	165	B.15	Issues of the first version of the odometry wheel . . . . .	207
6.22	Shopping run with other shoppers . . . .	167	B.16	The <i>force sensitive handle</i> . . . . .	208
6.23	Control shares during shopping run . . . .	167	B.17	Forces relevant for the <i>Manual Steering</i> <i>Mode</i> . . . . .	208
6.24	Commanding <i>InBOT</i> using a smartphone	168	B.18	Control chain of the <i>Manual Steering Mode</i>	209
6.25	Sketch of the user test: The intended path of the robot . . . . .	169	B.19	The three versions of the <i>force sensitive</i> <i>handle</i> . . . . .	210
7.1	Queuing up at the check out counter . . . .	173	B.20	Construction of the third version of the <i>force sensitive handle</i> . . . . .	211
7.2	The Virtual Train concept . . . . .	174	C.1	Example of a tracked object . . . . .	215
7.3	The Virtual Train concept . . . . .	175	C.2	User tracking by on-board camera: the color template . . . . .	216
7.4	The Virtual Train concept: Example 2 . . .	176	C.3	User tracking by onboard camera: ar- rangement of the directional antennas . . .	216
7.5	The Virtual Train concept: 5 simulated robots in one train . . . . .	176	C.4	User tracking by on-board camera: pro- cessing steps . . . . .	216
8.1	The robot <i>InBOT</i> . . . . .	179	C.5	<i>InBOT</i> Following a user by RFID and vi- sion . . . . .	217
8.2	The system <i>ETrolley</i> . . . . .	180	C.6	System architecture of the vision-based tracker . . . . .	218
8.3	The robot <i>Odete</i> . . . . .	180	C.7	Person tracking pipeline . . . . .	218
8.4	The walking robot <i>LAURON</i> . . . . .	181	C.8	Examples of the person tracker . . . . .	219
8.5	The robot <i>CityPod</i> . . . . .	182	C.9	Vision based person tracking: special sit- uations . . . . .	220
8.6	The robot <i>HoLLiE</i> . . . . .	183	C.10	The <i>InBOT-UI</i> . . . . .	221
B.1	<i>InBOT</i> 's equipment . . . . .	195	C.11	<i>InBOT-UI</i> login and recipes . . . . .	222
B.2	Hardware architecture of <i>InBOT</i> . . . . .	196	C.12	<i>InBOT-UI</i> main navigation screen . . . .	223
B.3	<i>InBOT2</i> design sketches . . . . .	197	C.13	Architecture of the <i>InBOT-UI</i> . . . . .	223
B.4	<i>InBOT2</i> construction . . . . .	197	C.14	The <i>CR-UI</i> . . . . .	224
B.5	Hardware architecture of <i>ETrolley</i> . . . .	199	C.15	IDS graph-based Navigation system . . .	225
B.6	Alternative <i>ETrolleys</i> . . . . .	200			
B.7	Electronic layout of <i>InBOT</i> . . . . .	201			
B.8	2D measurement wheel . . . . .	203			
B.9	Evaluation of odometry . . . . .	203			
B.10	Sketch of RFID barriers . . . . .	204			
B.11	Sketch of RFID reader . . . . .	204			
B.12	Sketch of RFID localisation . . . . .	205			





## Bibliography

- [1] R. Alami, R. Chatila, S. Fleury, M. Ghallab, and F. Ingrand. An architecture for autonomy. *International Journal of Robotics Research on Integrated Architectures for Robot Control and Programming*, 17:315–337, 1998.
- [2] J. Albiez. Verhaltensnetzwerke zur adaptiven Steuerung biologisch motivierter Laufmaschinen. *PhD Thesis, University of Karlsruhe (TH)*, 2006.
- [3] J. Albiez, T. Luksch, K. Berns, and R. Dillmann. A Behaviour Network Concept for Controlling Walking Machines. In: *Proc. 2nd International Symposium on Adaptive Motion of Animals and Machines*, 2003.
- [4] J. Albiez, T. Luksch, K. Berns, and R. Dillmann. An activation-based behavior control architecture for walking machines. *The International Journal on Robotics Research*, 22:203–211, 2003.
- [5] P. Althaus. Indoor Navigation for Mobile Robots: Control and Representations. *PhD Thesis, KTH Stockholm*, 2003.
- [6] P. Althaus and H. Christensen. Behaviour coordination for navigation in office environments. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2298–2304, 2002.
- [7] Apple-iTunes and EDEKA-Suedwest. Picture source: EDEKA App. <http://itunes.apple.com/de/app/edeka-sudwest/id391351382?mt=8>, 2012.
- [8] Apple-iTunes and Metro-Group. Picture source: real App MEA. <http://itunes.apple.com/de/app/real/id343799347?mt=8>, 2012.
- [9] R. C. Arkin and D. MacKenzie. Temporal Coordination of Perceptual Algorithms for Mobile Robot Navigation. *IEEE Transactions on Robotics and Automation*, 10(3):276 – 286, 1994.
- [10] C. Armbrust, M. Proetzsch, and K. Berns. Behaviour-Based Off-Road Robot Navigation. *KI - Künstliche Intelligenz, Springer Berlin / Heidelberg*, pages 155–160, 2011.
- [11] ATR, Technabob.com, and T. Kanda. Robovie-II. *Press Article, Online: http://technabob.com/blog/2009/12/16/robovie-ii-grocery-shopping-robot/*, 2012.
- [12] R. E. Bellmann. Dynamic Programming. *Princeton University Press*, 1957.
- [13] M. Bennewitz. Mobile Robot Navigation in Dynamic Environments. *PhD Thesis, Fakultät für Angewandte Wissenschaften, Albert-Ludwigs-Universität Freiburg im Breisgau*, 2004.
- [14] M. Bennewitz, W. Burgard, G. Cielniak, and S. Thrun. Learning motion patterns of people for compliant robot motion. *The International Journal of Robotics Research*, 24(1):31–48, 2005.
- [15] K. Berns. Webpage of ARTOS. <http://agrosy.informatik.uni-kl.de/roboter/artos/>, 2012.
- [16] K. Berns. Webpage of Marvin. <http://agrosy.informatik.uni-kl.de/roboter/marvin/>, 2012.
- [17] K. Berns. Webpage of RAVON. <http://agrosy.informatik.uni-kl.de/roboter/ravon/>, 2012.
- [18] K. Berns and S. A. Mehdi. Use of an Autonomous Mobile Robot for Elderly Care. *IEEE Computer Society - Advanced Technologies for Enhancing Quality of Life (AT-EQUAL)*, pages 121–126, 2010.
- [19] M. Bluemel. Optimierung des taktischen Verhaltens in Simulationen von Fußgängern. 2008.
- [20] C. Bogdan, D. Ertl, H. Huettenrauch, M. Goeller, A. Green, K. S. Eklundh, J. Falb, and H. Kaindl. Evaluation of Robot Body Movements Supporting Communication: Towards HRI on the Move. *New Frontiers in Human-Robot Interaction*, pages 185–210, 2011.
- [21] C. Bogdan and M. Goeller. Towards a Framework for Design and Evaluation of Mixed Initiative Systems: Considering Movement as a Modality. In: *Proc. Workshop on Improving Human-Robot Communication with Mixed-Initiative and Context-Awareness of the 18th IEEE International Symposium on Robot and Human Interactive Communication (RoMan), Toyama, Japan, Oct., 2009*.
- [22] R. Bonasso, D. Kortenkamp, D. Miller, R. J. Firby, E. Gat, and M. G. Slack. Experiences with an architecture for intelligent, reactive agents. *Journal of experimental and Theoretical Artificial Intelligence*, 9:237–256, 1996.
- [23] J. Borenstein and L. Feng. UMBmark : A Benchmark Test for Measuring Odometry Errors in Mobile Robots. In: *Proc. SPIE Conference on Mobile Robots*, 1995.
- [24] J. Borenstein and Y. Koren. Real-time Obstacle Avoidance for Fast Mobile Robots in Cluttered Environments. In: *Proc. IEEE International Conference on Robotics and Automation (ICRA), Cincinnati, Ohio (USA)*, pages 572–577, 1990.
- [25] J. Borenstein and Y. Koren. The vector field histogram - fast obstacle avoidance for mobile robots. *IEEE Journal of Robotics and Automation*, 7(3):278–288, 1991.

- [26] A. Bosien, M. Venzke, and V. Turau. A rewritable RFID environment for AGV navigation. *In: Proc. 5th Int. Workshop on Intelligent Transportation (WIT'08). Hamburg, Germany, 2008.*
- [27] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. *IEEE Journal of Robotics and Automation*, 2(1):14–23, 1986.
- [28] R. A. Brooks. Intelligence without representation. *Artificial Intelligence*, 47:139–159, 1991.
- [29] J. J. Bryson. The Behavior-Oriented Design of Modular Agent Intelligence. *Agent Technologies, Infrastructures, Tools, and Applications for e-Services*, Springer, pages 61–76, 2003.
- [30] J. Buhmann, W. Burgard, A. B. Cremers, D. Fox, T. Hofmann, F. Schneider, J. Strikos, and S. Thrun. The Mobile Robot Rhino. *The AI Magazine*, 16, 1996.
- [31] W. Burgard, A. B. Cremers, D. Fox, D. Haehnel, G. Lakemeyer, D. Schulz, W. Steiner, and S. Thrun. Experiences With An Interactive Museum Tour-Guide Robot. *Artificial Intelligence - Special issue on applications of artificial intelligence*, 114(1-2):3–55, 1998.
- [32] D. Castro, U. Nunes, and A. Ruano. Reactive local navigation. *In: Proc. IEEE 28th Annual Conference of the Industrial Electronics Society (IECON)*, 3:2427–2432, 2002.
- [33] D. G. Çakir and A. Çakir. Neufassung DIN 33402 Aktualisierte Körpermaße - Auswirkungen auf die Produktgestaltung von Büromöbeln und die Arbeitsplatzgestaltung im Büro- und Verwaltungsbereich. *Report, ERGONOMIC Institut für Arbeits- und Sozialforschung Forschungsgesellschaft mbH*, 2006.
- [34] R. Chatila and J. Laumond. Position referencing and consistent world modeling for mobile robots. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2:138–145, 1985.
- [35] H. Choset and J. Burdick. Sensor based planning. I. The generalized Voronoi graph. *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, 2:1649–1655, 1995.
- [36] H. Choset and J. Burdick. Sensor-Based Exploration: The Hierarchical Generalized Voronoi Graph. *The international journal of robotics research*, 2(19):96–125, 2000.
- [37] A. Codas and M. Devy. RFID-based robot localization and object mapping in a changing environment. *LAAS Report*, 2010.
- [38] DFKI. Webpage of Innovative retail laboratory project. <http://www.innovative-retail.de>, 2012.
- [39] D. Dieckmann. Die Feuersicherheit in Theatern. *PhD Thesis, Technische Hochschule Hannover*, 1911.
- [40] EDEKA-Suedwest and Burda-Digital-Systems. Webpage: EDEKA Suedwest App. <https://www.edeka.de/SUEDWEST/Content/de/SuedwestApp.html>, 2013.
- [41] D. Ertl, J. Falb, and H. Kaindl. Semi-automatically Configured Fission for Multimodal User Interfaces. *Third International Conference on Advances in Computer-Human Interactions (ACHI '10)*, pages 85–90, 2010.
- [42] ETHZ. Online picture gallery of Robox. <http://projects.asl.ethz.ch/robox/>, 2011.
- [43] P. Fiorini and Z. Shillert. Motion Planning in Dynamic Environments using Velocity Obstacles. *International Journal of Robotics Research*, 17, 1998.
- [44] H. Fischer. Die Leistungsfähigkeit von Türen, Gängen und Treppen bei ruhigem, dichtem Verkehr. *Ph.D. Thesis, Technische Hochschule Dresden*, 1933.
- [45] Forschungszentrum Informatik (FZI). Webpage: Modular Controller Architecture Version 2 (MCA2). <http://www.mca2.org/>, 2011.
- [46] D. Fox, W. Burgard, and S. Thrun. Controlling synchro-drive robots with the dynamic window approach to collision avoidance. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS-96)*, 3:1280–1287, 1996.
- [47] D. Fox, W. Burgard, and S. Thrun. The Dynamic Window Approach to Collision Avoidance. *IEEE Robotics and Automation Magazine*, 4(1):23–33, 1997.
- [48] T. Fraichard. Trajectory Planning in Dynamic Workspace: a State-Time Space Approach. *Advanced Robotics*, 13(6), 1998.
- [49] Fraunhofer-IPA. Webpage: Care-O-Bot. <http://www.care-o-bot.de/>, 2010.
- [50] Fraunhofer-IPA. Picture source: RACOON. Webpage: *Serviceroboter Anwendungen*, [www.ipa.fraunhofer.de](http://www.ipa.fraunhofer.de), 2011.
- [51] K. Fujimura and H. Samet. A hierarchical strategy for path planning among moving obstacles. *IEEE Transaction on Robotics and Automation*, 5(1), 1989.
- [52] Fujitsu. Fujitsu Begins Limited Sales of Service Robot "enon" for Task Support in Offices and Commercial Establishments. *Press release*, <http://www.fujitsu.com/global/news/pr/archives/month/2005/20050913-01.html>, 2005.
- [53] Fujitsu. Fujitsu Introduces Wireless Mobile Attendant for U-Scan Self-checkout. *Press release*, [http://www.fujitsu.com/us/news/pr/ftxs\\_20050213-05.html](http://www.fujitsu.com/us/news/pr/ftxs_20050213-05.html), 2005.

- [54] Fujitsu and Gizmag. U-Scan shopper. *Article*, <http://www.gizmag.com/go/4345/>, 2012.
- [55] C. Fulgenzi, A. Spalanzani, and C. Laugier. Dynamic Obstacle Avoidance in uncertain environment combining PVOs and Occupancy Grid. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2007.
- [56] B. Gassmann. Modellbasierte, sensorgestützte Navigation von Laufmaschinen im Gelände. *PhD Thesis, Fakultät für Informatik, Universität Karlsruhe (TH)*, 2007.
- [57] E. Gat. On three-layer architectures. *Artificial Intelligence and Mobile Robots: Case Studies of Successful Robot Systems*, AAAI Press, 1998.
- [58] L. Gemzoe. Are Pedestrians Invisible in the Planning Process? Copenhagen as a Case Study. *In: Proc. Walk21 Conference, Perth, Australien*, 2001.
- [59] T. Germa, F. Lerasle, N. Ouadah, V. Cadenat, and M. Devy. Vision and RFID-based person tracking in crowds from a mobile robot. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5591–5596, 2009.
- [60] D. F. Glas, T. Kanda, H. Ishiguro, and N. Hagita. Field Trial for Simultaneous Teleoperation of Mobile Social Robots. *In: Proc. 4th ACM/IEEE international conference on Human robot interaction (HRI)*, pages 149–156, 2009.
- [61] M. Goeller, T. Kerscher, M. Ziegenmeyer, A. Roennau, J. Zoellner, and R. Dillmann. Haptic control for the interactive behavior operated shopping trolley InBOT. *In: Proc. New Frontiers in Human-Robot Interaction Workshop, Convention Artificial Intelligence and Simulation of Behaviour (AISB), Edinburgh (UK)*, 2009.
- [62] M. Goeller, A. Roennau, A. Gorbunov, G. Heppner, and R. Dillmann. Pushing Around a Robot: Force-Based Manual Control of the Six-Legged Walking Robot LAURON. *In: Proc. IEEE International Conference on Robotics and Biomimetics (RoBio), Phuket, Thailand*, 2011.
- [63] M. Goeller, F. Steinhardt, T. Kerscher, R. Dillmann, M. Devy, T. Germa, and F. Lerasle. Sharing of control between an interactive shopping robot and its user in collaborative tasks. *In: Proc. 19th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN), Viareggio, Italy*, pages 626–631, Sept. 2010.
- [64] M. Goeller, F. Steinhardt, T. Kerscher, J. Zoellner, and R. Dillmann. RFID Transponder Barriers as Artificial Landmarks for the Semantic Navigation of Autonomous Robots. *In: Proc. 11th International Conference on Climbing and Walking Robots and the Support Technologies for Mobile Machines (CLAWAR), Coimbra, Portugal*, Sept. 2008.
- [65] P. G. Griffiths and R. B. Gillespie. Sharing control between humans and automation using haptic interface: Primary and secondary task performance benefits. *Human Factors, the Journal of the Human Factors and Ergonomics Society*, 47(3):574–590, 2005.
- [66] H. M. Gross, H.-J. Boehme, C. Schroeter, S. Mueller, A. Koenig, E. Einhorn, C. Martin, M. Merten, and A. Bley. TOOMAS: Interactive Shopping Guide robots in everyday use - final implementation and experiences from long-term field trials. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2005–2012, 2009.
- [67] H. M. Gross, H.-J. Boehme, C. Schroeter, S. Mueller, A. Koenig, C. Martin, M. Merten, and A. Bley. Shop-Bot: Progress in developing an interactive mobile shopping assistant for everyday use. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 3471–3478, 2008.
- [68] D. Haehnel, W. Burgard, D. Fox, K. Fishkin, and M. Philipose. Mapping and localization with RFID technology. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA), New Orleans, LA (USA)*, pages 1015–1020, Apr. 2004.
- [69] M. A. Hearst, J. Allen, C. Guinn, and E. Horvtz. Mixed-initiative interaction. *Trends and Controversies, IEEE Intelligent Systems*, 14(5):14–23, Sept. 1999.
- [70] A. Hermann, Z. Xue, S. W. Ruhl, and R. Dillmann. Hardware and software architecture of a bimanual mobile manipulator for industrial application. *In: Proc. IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2282–2288, Dec. 2011.
- [71] F. Hoeller, D. Schulz, M. Moors, and F. Schneider. Accompanying persons with a mobile robot using motion prediction and probabilistic roadmaps. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1260–1265, 2007.
- [72] K. Homma and E. Takenaka. An image processing method for feature extraction of space-occupying lesions. *Journal of nuclear medicine, Society of Nuclear Medicine*, 26(12):1472–1477, Dec. 1985.
- [73] S. P. Hoogendoorn, P. H. L. Bovie, and W. Daamen. Microscopic pedestrian wayfinding and dynamics modelling. *Pedestrian and Evacuation Dynamics, Springer, Berlin*, pages 123–154, 2002.
- [74] B. K. P. Horn. Robot Vision. *The MIT Press*, 1986.
- [75] R. A. Howard. Dynamic Programming and Markov Processes. *MIT Press, Cambridge*, 1960.

- [76] H. Hu, M. Brady, and P. Probert. Navigation and control of a mobile robot among moving obstacles. *In: Proc. 30th IEEE Conference on Decision and Control, Brighton, UK*, pages 698–703, 1991.
- [77] H.-P. Huang and S.-Y. Chung. Dynamic visibility graph for path planning. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3:2813–2818, 2004.
- [78] H. Huettneraich, B. Cristian, A. Green, K. S. Eklundh, D. Ertl, J. Falb, H. Kaindl, and M. Goeller. Evaluation of Robot Body Movements Supporting Communication. *In: Proc. 2010 Convention Artificial Intelligence and Simulation of Behaviour (AISB2010), Leicester, UK*, 2010.
- [79] IBM-Research-Solutions. IBM Personal Shopping Assistant. *Solution-Sheet, available online: [http://www.xr23.com/studies/PSA\\_solution\\_sheet.pdf](http://www.xr23.com/studies/PSA_solution_sheet.pdf)*, 2012.
- [80] IBM-Research-Solutions. Personal Shopping Assistant. *Webpage, <http://domino.research.ibm.com/odis/odis.nsf/pages/solution.10.html>*, 2012.
- [81] IRobot. Picture Source: Roomba. *<http://www.irobot.com>*, 2012.
- [82] H. Ishiguro, T. Kanda, K. Kimoto, and T. Ishida. A Robot Architecture Based on Situated Modules. *In: Proc. International Conference on Intelligent Robots and Systems (IROS)*, 3:1617–1624, 1999.
- [83] H. Jaeger and T. Christaller. Dual Dynamics: Designing Behavior Systems for Autonomous Robots. *Artificial Life and Robotics*, 2:76–79, 1998.
- [84] J. Janet, R. Luo, and M. Kay. The essential visibility graph: an approach to global motion planning for autonomous mobile robots. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2:1958–1963, 1995.
- [85] T. Kanda, H. Ishiguro, T. Ono, M. Imai, and R. Nakatsu. Development and Evaluation of an Interactive Humanoid Robot "Robovie". *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 2:1848–1855, 2002.
- [86] T. Kanda, M. Shiomi, Z. Miyashita, H. Ishiguro, and N. Hagita. An Affective Guide Robot in a Shopping Mall. *In: Proc. International Conference in Human-Robot Interaction (HRI)*, 2009.
- [87] S. Kavaldjian, D. Raneburger, and J. Falb. Semi-automatic user interface generation considering pointing granularity. *IEEE International Conference on Systems, Man and Cybernetics (SMC)*, pages 2052–2058, 2009.
- [88] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 1(5):90–98, 1986.
- [89] O. Khatib and R. Chatila. An extended potential field approach for mobile robot sensor-based motions. *In: Proc. International Conference on Intelligent Autonomous Systems (IAS-4)*, 1995.
- [90] J. Kim, M.-T. Choi, M. Kim, S. Kim, M. Kim, S. Park, J. Lee, and B. Kim. Intelligent Robot Software Architecture. *Recent Progress in Robotics: Viable Robotic Service to Human*, Springer, pages 385–397, 2008.
- [91] K. Kim and L. S. Davis. Multi-camera Tracking and Segmentation of Occluded People on Ground Plane Using Search-Guided Particle Filtering. *In: Proc. European Conference on Computer Vision (ECCV)*, pages 98–109, 2006.
- [92] M. Kim, S. Kim, S. Park, M.-T. Choi, M. Kim, and H. Goma. UML-based service robot software development: a case study. *In: Proc. 28th international conference on software engineering*, 2006.
- [93] Klever-Marketing and Gizmag. Klever Cart. *Article, <http://www.gizmag.com/go/3751/>*, 2012.
- [94] Klever-Marketing and Time-Domain. Giving Cart. *Press Release, <http://www.timedomain.com/news/giving-cart.php>*, 2012.
- [95] S. Knoop, S. R. Schmidt-Rohr, and R. Dillmann. A Flexible Task Knowledge Representation for Service Robot. *In: Proc. 9th International Conference on Intelligent Autonomous Systems (IAS-9)*, 2006.
- [96] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, pages 1398–1404, 1991.
- [97] T. Kretz. Pedestrian Traffic - Simulation and Experiments. *PhD Thesis, Fachbereich Physik, Universität Duisburg-Essen*, 2007.
- [98] B. Krogh. A generalized potential field approach to obstacle avoidance control. *SME Conference Proceedings on Robotics Research: The Next five Years and Beyond*, 1984.
- [99] E. Kruse and F. Wahl. Camera-Based Monitoring System for Mobile Robot Guidance. *In: Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)*, (October), 1998.
- [100] B. J. Kuipers, J. Modayil, P. Beeson, M. MacMahon, and F. Savelli. Local metrical and global topological maps in the hybrid spatial semantic hierarchy. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 5:4845–4851, 2004.
- [101] S. Y. T. Lang and B. Y. Chee. Coordination of behaviours for mobile robot floor cleaning. *In: Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2:1236–1241, 1998.

- [102] D. Langer, J. K. Rosenblatt, and M. Hebert. A behavior-based system for off-road navigation. *IEEE Transactions on Robotics and Automation*, 10(6):776–783, 1994.
- [103] F. Large, D. Vasquez, T. Fraichard, and C. Laugier. Avoiding cars and pedestrians using velocity obstacles and motion prediction. In: *Proc. IEEE Intelligent Vehicles Symposium*, pages 375–379, 2004.
- [104] M. Littman, A. Cassandra, and L. P. Kaelbling. Learning policies for partially observable environments: Scaling up. In: *Proc. 12th International Conference on Machine Learning*, 1995.
- [105] L. Lulu and A. Elnagar. A comparative study between visibility-based roadmap path planning algorithms. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 3263–3268, 2005.
- [106] C. Manresa-yea, J. Varona, R. Mas, and F. J. Perales. Hand Tracking and Gesture Recognition for Human-Computer Interaction. *Electronic Letters on Computer Vision and Image Analysis*, 5(3):96–104, 2005.
- [107] M. Mataric. Behaviour-based control: Examples from navigation, learning, and group behaviour. *Journal of Experimental and Theoretical Artificial Intelligence*, 9(2-3):323–336, 1997.
- [108] N. Matsuhira, F. Ozaki, S. Tokura, T. Sonoura, and T. Tasaki. Development of robotic transportation system - Shopping support system collaborating with environmental cameras and mobile robots -. In: *Proc. 41st International Symposium on Robotics (ISR) and 6th German Conference on Robotics (ROBOTIK)*, 2010.
- [109] S. A. Mehdi and K. Berns. Ordering of Robotic Navigational Tasks in Home Environment. *Trends in Intelligent Robotics, Springer Berlin Heidelberg*, pages 242–249, 2010.
- [110] M. Mehmood, L. Kulik, and E. Tanin. Autonomous navigation of mobile agents using RFID-enabled space partitions. In: *Proc. 16th ACM SIGSPATIAL international conference on advances in geographic information systems*, 2008.
- [111] Mercatus Solutions. Springbord cart. *Webpage*, <http://www.mercatustechnologies.com/solutions/cart-solution/>, 2012.
- [112] Metralabs. SCITOS A5 - autonomous mobile robot system for interaction and guidance. *Product Flyer*, 2012.
- [113] Metro-Group. Future Store Project. *Webpage*, <http://www.future-store.org/fsi-internet/html/en/20118/index.html>, 2013.
- [114] Metro-Group. Real App MEA. *Webpage*, <http://www.future-store.org/fsi-internet/html/de/25891/index.html>, 2013.
- [115] A. Millonig and K. Schechter. Decision Loads and Route Qualities for Pedestrians - Key Requirements for Design of Pedestrian Navigation Series. *Pedestrian and Evacuation Dynamics, Springer*, 1:109–118, 2006.
- [116] J. Modayil, P. Beeson, and B. J. Kuipers. Using the topological skeleton for scalable global metrical map-building. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2:1530–1536, 2004.
- [117] S. Mueller, E. Schaffernicht, A. Scheidig, H.-J. Boehme, and H. M. Gross. Are you still following me? In: *Proc. of the 3rd European Conference on Mobile Robots (ECMR)*, pages 211–216, 2007.
- [118] I. A. D. Nesnas, R. Simmons, D. Gaines, C. Kunz, A. Diaz-Calderon, T. Estlin, R. Madison, J. Guineau, M. McHenry, I.-H. Shu, and D. Apfelbaum. CLARATy: Challenges and Steps toward Reusable Robotic Software. *International Journal of Advanced Robotic Systems*, 3(1), 2006.
- [119] A. Newell and H. Simon. GPS: A program that simulates human thought. *AAI Press / MIT Press*, 1995.
- [120] J. Oberlaender, K. Uhl, J. Zoellner, and R. Dillmann. A Region-based SLAM Algorithm Capturing Metric, Topological, and Semantic Properties. In: *Proc. IEEE International Conference of Robotics and Automation (ICRA)*, 2008.
- [121] M. Ocana, L. Bergasa, and M. Sotelo. Indoor robot navigation using a POMDP based on WiFi and ultrasound observations. In: *Proc. IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2592–2597, 2005.
- [122] E. Owen, U. Valle, and L. Montano. A robocentric motion planner for dynamic environments using the velocity space. In: *Proc. IEEE International Conference on Intelligent Robots and Systems (IROS)*, 2006.
- [123] P. Pirjanian. Behavior Coordination Mechanisms - State-of-the-art. *Tech Report, University of Southern California*, 1999.
- [124] T. Priya and K. Sridharan. An efficient algorithm to construct reduced visibility graph and its FPGA implementation. In: *Proc. 17th International Conference on VLSI Design*, pages 1057–1062, 2004.
- [125] M. Proetzsch, T. Luksch, and K. Berns. Development of complex robotic systems using the behavior-based control architecture iB2C. *Journal Robotics and Autonomous Systems*, 58(1), 2010.
- [126] Putzmeister. Picture Source: Skywash. <http://pmw.de>, 2012.
- [127] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In: *Proc. IEEE International Conference on Robotics and Automation*, 2:802–807, 1993.

- [128] Y. Raoui, M. Goeller, M. Devy, T. Kerscher, J. Zoellner, R. Dillmann, and A. Coustou. RFID-based topological and metrical self-localization in a structured environment. *In: Proc. International Conference on Advanced Robotics (ICAR), Munich, Germany, 2009.*
- [129] R. Regele. Kooperative Multi-Roboter-Wegplanung durch heuristische Prioritätsanpassung. *PhD Thesis, Fakultät für Informatik, Elektrotechnik und Informationstechnik, Universität Stuttgart, 2007.*
- [130] K. Regenstern, T. Kerscher, C. Birkenhofer, T. Asfour, J. Zoellner, and R. Dillmann. Universal Controller Module (UCoM) - component of a modular concept in robotic systems. *In: Proc. IEEE International Symposium on Industrial Electronics (ISIE), 2007.*
- [131] RFID Journal, Klever-Marketing, and Time-Domain. Picture source: Giving Cart. <http://www.rfidjournal.com/article/print/5248>, 2012.
- [132] Robostore. Picture source: Metropolis - Die Geschichte der Roboter in Literatur und Film. [http://www.robotstore.de/roboter\\_in\\_filmen.htm](http://www.robotstore.de/roboter_in_filmen.htm), 2010.
- [133] Robowatch. Picture Source: MOSRO. <http://www.robowatch.de>, 2012.
- [134] J. K. Rosenblatt. DAMN: A Distributed Architecture for Mobile Navigation. *AAAI Spring Symposium on Lessons Learned from Implemented Software Architectures for Physical Agents, Stanford, CA. AAAI Press, 1995.*
- [135] R. Rothenberg and CMU. Real Estate Robots. *Press release, http://www.cs.cmu.edu/~minerva/press/realprogress/index.html*, 2011.
- [136] A. Saotti. The Uses of Fuzzy Logic in Autonomous Robot Navigation: a catalogue raisonne. *Technical Report v2.1, IRIDIA, Universite Libre de Bruxelles, Brussels, Belgium, 1997.*
- [137] T. Sattel and T. Brandt. Ground vehicle guidance along collision-free trajectories using elastic bands. *In: Proc. American Control Conference, 7:4991–4996, 2005.*
- [138] A. Schadschneider, W. Klingsch, H. Kluepfel, T. Kretz, C. Rogsch, and A. Seyfried. Evacuation Dynamics: Empirical Results, Modeling and Applications. *Encyclopedia of Complexity and System Science, 5(57):3142–3176, 2008.*
- [139] B. H. Schaefer and K. Berns. Ravon - an autonomous vehicle for risky intervention and surveillance. *In: Proc. International Workshop on Robotics for risky intervention and environmental surveillance (RISE), 2006.*
- [140] M. Scheutz and V. Andronache. Architectural mechanisms for dynamic changes of behavior selection strategies in behavior-based systems. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 34(6):2377–2395, 2004.*
- [141] D. Schmidt, T. Luksch, J. Wettach, and K. Berns. Autonomous Behavior-Based Exploration of Office Environments. *In: Proc. 3rd International Conference on Informatics in Control, Automation and Robotics (ICINCO), Portugal, pages 235–240, 2006.*
- [142] K. Schneider and T. Schuele. Averest: Specification, Verification, and Implementation of Reactive Systems. *In: Proc. Conference on Application of Concurrency to System Design (A CSD), St. Malo, France, 2005.*
- [143] K. U. Scholl, J. Albiez, and B. Gassmann. MCA - An Expandable Modular Controller Architecture. *In: Proc. 3rd Real-Time Linux Workshop, 2001.*
- [144] K. U. Scholl, M. Klein, and B. Gassmann. Zentrale Aufgabenverteilung in einem fahrerlosen Transportsystem. *In: Proc. Autonome Intelligente Systeme (AMS), Springer, pages 253–259, 2005.*
- [145] M. J. Schoppers. Universal Plans for Reactive Robots in Unpredictable Environments. *In: Proc. 10th international joint conference on Artificial intelligence (IJCAI), 2, 1987.*
- [146] A. Seyfried, B. Steffen, W. Klingsch, and M. Boltes. The Fundamental Diagram of Pedestrian Movement Revisited. *Journal of Statistical Mechanics, (10), 2005.*
- [147] T. B. Sheridan. Telerobotics, automation, and human supervisory control. *MIT Press, 1992.*
- [148] T. B. Sheridan and W. L. Verplank. Human and Computer Control of Undersea Teleoperation. *Report, Man-Machine Systems Lab, MIT, 1978.*
- [149] R. Siegart, K. Arras, S. Bouabdallah, D. Burnier, G. Froidevaux, X. Greppin, B. Jensen, A. Lorotte, L. Mayor, M. Meisser, R. Philippsem, R. Piguat, G. Ramel, G. Terrien, and N. Tomatis. Robox at Expo.02: A large-scale installation of personal robots. *Robotics and Autonomous Systems, Elsevier, 42(3-4):203–222, 2003.*
- [150] Siemens. Picture source: Die vollautomatische fahrerlose U-Bahn. *Rubrik Verkehr, www.siemens.de*, 2011.
- [151] K.-T. Song and C. Chang. Reactive navigation in dynamic environment using a multisensor predictor. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics, 29(6):870–880, 1999.*
- [152] M. Spaan and N. Vlassis. A point-based POMDP algorithm for robot planning. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA), 3:2399–2404, 2004.*

- [153] C. Stachniss and W. Burgard. An Integrated Approach to Goal-directed Obstacle Avoidance under Dynamic Constraints for Dynamic Environments. *In: Proc. of IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2002.
- [154] L. Steels. A case study in the behaviour-oriented design of autonomous agents. *In: Proc. 3rd international conference on Simulation of adaptive behavior: from animals to animats (SAB94)*, pages 445–452, 1994.
- [155] F. Steinhardt, M. Strand, and J. Zoellner. Autonomous Navigation of a Personal Transporter within Moving Human Groups using Reactive Control. *In: Proc. 12th International Conference on Intelligent Autonomous Systems (IAS 12)*, 2012.
- [156] A. Stentz. The focussed D\* algorithm for real-time replanning. *In: Proc. International Joint Conference on Artificial Intelligence (IJCAI-95), Montreal, Quebec*, 1995.
- [157] Swisslog Healthcare Solutions. Telelift (Image source). *Swisslog Accessories, Automated Guided Vehicle (AGV) Bulk Material Carts*, 2011.
- [158] S. Thrun. To Know or Not To Know: On the Utility of Models in Mobile Robotics. *AI Magazine*, 18:47–54, 1997.
- [159] S. Thrun. Learning metric-topological maps for indoor mobile robot navigation. *Artificial Intelligence*, 99(1):21–71, 1998.
- [160] S. Thrun. Learning Occupancy Grids With Forward Sensor Models. *Article, Carnegie Mellon University*, 2002.
- [161] S. Thrun. Robotic Mapping: A Survey. *Exploring Artificial Intelligence in the New Millenium, Morgan Kaufmann*, 2002.
- [162] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dallaert, D. Fox, D. Haehnel, G. Lakemeyer, C. Rosenberg, N. Roy, N. Schulte, J. Schulte, and W. Steiner. Experiences with two Deployed Interactive Tour-Guide Robots. *In: Proc. International Conference on Field and Service Robotics*, 1999.
- [163] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Haehnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: a second-generation museum tour-guide robot. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, 3:1999–2005, 1999.
- [164] S. Thrun, W. Burgard, and D. Fox. Probabilistic Robotics (Intelligent Robotics and Autonomous Agents). *MIT Press*, 2005.
- [165] S. Thrun, J. Gutmann, D. Fox, W. Burgard, and B. J. Kuipers. Integrating topological and metric maps for mobile robot navigation: A statistical approach. *In: Proc. AAAI 15th National Conference on Artificial Intelligence*, 1998.
- [166] S. Tokura, T. Sonoura, T. Tasaki, N. Matsuhira, M. Sano, and K. Komoriya. Robotic transportation system for shopping support services. *In: Proc. 18th IEEE International Symposium on Robot and Human Interactive Communication (RO-MAN)*, page 320, 2009.
- [167] TU Ilmenau. Picture source: TOOMAS. *Robotergalerie*, <http://www.tu-ilmenau.de/fakia/2203+M52087573ab0.0.html>, 2011.
- [168] K. Uhl, B. Gassmann, and J. Oberlaender. Modular Controller Architecture Version 2. [www.mca2.org](http://www.mca2.org), 2010.
- [169] K. Uhl, A. Roennau, and R. Dillmann. From Structure to Actions: Semantic Navigation Planning in Office Environments. *In: Proc. IROS Workshop on Perception and Navigation for Autonomous Vehicles in Human Environment*, 2011.
- [170] I. Ulrich and J. Borenstein. VFH+: reliable obstacle avoidance for fast mobile robots. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA), Leuven, Belgium*, 2:1572–1577, 1998.
- [171] I. Ulrich and J. Borenstein. VFH \*: Local Obstacle Avoidance with Look-Ahead Verification. *In: Proc. IEEE International Conference on Robotics and Automation (ICRA)*, (April):2505–2511, 2000.
- [172] University of Bonn. Picture source: Rhino. *Image gallery*, <http://www.iai.uni-bonn.de/~rhino/tourguide/html/gallery.html>, 2011.
- [173] VDMA. World Robotics 2008. *Report, IFR Statistical Department, VDMA Robotics + Automation association*, 2008.
- [174] P. Vorst, S. Schneegans, B. Yang, and A. Zell. Self-Localization with RFID snapshots in densely tagged environments. *In: Proc. the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1353–1358, 2008.
- [175] P. Vorst, J. Sommer, C. Hoene, P. Schneider, C. Weiss, T. Schairer, W. Rosenstiel, A. Zell, and G. Carle. Indoor positioning via three different RF technologies. *In: Proc. 4th European Workshop on RFID Systems and Technologies (RFID SysTech 08), Freiburg, Germany*, 2008.
- [176] P. Vorst and A. Zell. Semi-autonomous learning of an RFID sensor model for mobile robot self-localization. *European Robotics Symposium, Springer Tracts in Advanced Robotics*, 44:273–282, 2008.

## Bibliography

---

- [177] U. Weidmann. Transporttechnik der Fußgänger - Transporttechnische Eigenschaften des Fußgängerverkehrs. *Schriftenreihe des Institut für Verkehrsplanung, Transporttechnik, Strassen- und Eisenbahnbau der ETH Zürich*, (90), 1993.
- [178] Wikimedia Commons. Picture source: Unimate. <http://commons.wikimedia.org/wiki/File:Unimate.jpg>, 2010.
- [179] M. Ziegenmeyer. Entwicklung einer semantischen Missionssteuerung für autonome Inspektionsroboter. *PhD Thesis, Fakultät für Informatik, Karlsruhe Institut für Technologie (KIT)*, 2011.
- [180] M. Ziegenmeyer, K. Uhl, S. Sayler, J. Zoellner, and R. Dillmann. A Semantic Approach for the Inspection of Complex Environments with Autonomous Service Robots. *In: Proc. IARP Workshop on Environmental Maintenance & Protection, Baden-Baden, Germany*, 2008.