Intuitive Human-Robot Cooperation

zur Erlangung des akademischen Grades eines

Doktors der Ingenieurwissenschaften

der Fakultät für Informatik
der Universität Fridericiana zu Karlsruhe (TH)

genehmigte

D i s s e r t a t i o n

von

Dipl.–Ing. Andreas Schmid

aus Stuttgart

Tag der mündlichen Prüfung:	18. Juli 2008
Erster Gutachter:	Prof. Dr.-Ing. Heinz Wörn
Zweiter Gutachter:	Prof. Dr.-Ing. Uwe D. Hanebeck

Zusammenfassung

Die hier vorgestellte Arbeit entstand im Rahmen des Sonderforschungsbereiches 588 „Humanoide Roboter – Lernende und kooperierende multimodale Roboter“ im Teilprojekt K2, das sich mit der Mensch-Roboter Kooperation beschäftigt. Dabei wurde eine haptische Schnittstelle entworfen, die dem Benutzer mit Hilfe einer Taktilen Sprache eine weitere non-verbale Interaktionsmodalität zur Verfügung stellt. Außerdem wurde eine Methode zur proaktiven Planung und Ausführung von Roboterhandlungen auf Basis der geschätzten Intention des Menschen erforscht. Um beide Methoden in die bestehende einfache Robotersteuerung zu integrieren, wurde eine adäquate Roboterarchitektur konzipiert und implementiert, die die bisherige Steuerung und neu benötigte Architekturkomponenten umfasst und zu einem Gesamtsystem verbindet.

Die Grobstruktur der Roboterarchitektur besteht aus drei Ebenen, wobei die oberen beiden Ebenen bestehend aus Aufgabenplaner und Ausführungsüberwachung die wesentlich neuen Beiträge darstellen. Der Aufgabenplaner dient dazu, die nächste auszuführende Aufgabe auszuwählen. Er wurde so gestaltet, dass der Roboter sowohl Befehle von sämtlichen Mensch-Maschine-Schnittstellen als auch dem Modul für proaktives Handeln erhalten kann. Dazu wurde ein Algorithmus definiert, der in jeder Situation den passenden Befehl selektiert. Wurde ein Kommando ausgewählt, sorgt die Ausführungsüberwachung für die korrekte Abarbeitung der Aufgabe.

Mit „Taktiler Sprache“ wird eine in dieser Arbeit entwickelte Mensch-Roboter Schnittstelle bezeichnet, die es dem Menschen ermöglicht, den Roboter über seine künstliche Haut oder ein Touchpad zu steuern. Das Verarbeitungssystem der taktilen Sprache lässt sich in drei Abschnitte einteilen. Zum einen umfasst es die Sensorschnittstellen und die Bildverarbeitung, zum anderen die Erkennung der Vokabeln der Sprache, und schließlich die Wortinterpretation, also das Sprachsystem selbst und die Anbindung an die Robotersteuerung. Für die taktile Sprache werden verschiedene Eingabemodi angeboten: Der indirekte Modus, mit dem eine Eingabe von Richtungsangaben (Vektoren) und Winkeln (Kreisbögen) möglich ist; der direkte Modus, mit dem eine echtzeitfähige Steuerung des Roboters erfolgen kann; die Knopffunktion, mit der gezielt Bereiche des taktilen Sensors eine knopfähnliche Funktion zugeschrieben werden kann, um Befehle an den Roboter direkt per Fingerberührung aufrufen zu können; und schließlich der normale Modus, in dem mit Hilfe des integrierten frei verfügbaren Handschrifterkenners Merlin und der selbst entworfenen Symbolerkennung verschiedene Buchstaben oder Symbole eingegeben werden können. Die taktile Sprache ist sehr ausdrucksstark, da während der Symboleingabe unterschiedliche Eigenschaften wie Druck, Pfadlänge und Geschwindigkeit berücksichtigt wer-

den, und somit eine Vielzahl von Variationen mit einer Eingabe eines Symbols möglich sind. Sie ist darüber hinaus auch echtzeitfähig, weil mit ihr sowohl eine Direktsteuerung des Roboters betrieben werden kann, als auch eine sofortige Erkennung der Symbole nach der Eingabe vorliegt. Die Interpretation erfolgt schließlich mit Hilfe eines Zustandsautomaten, der die taktile Sprache definiert und überprüft, ob die taktile Eingabe einen Sinn ergibt oder nicht. Sollte eine gültige Eingabe vorliegen, werden die Daten an die Robotersteuerung übertragen.

Ein traditioneller Serviceroboter wird ausschließlich durch explizite Befehle des Menschen gesteuert. Das in dieser Arbeit neu begründete Paradigma des proaktiven Handelns stellt dem eine implizite Kommandierung des Roboters gegenüber, die eine intuitivere Mensch-Roboter Kooperation ermöglichen soll. Ausgehend von der geschätzten Intention des menschlichen Kooperationspartners wählt der Roboter eine dazu passende Handlung aus und beginnt diese auszuführen, falls der Mensch keinen expliziten Befehl geäußert hat. Dabei wird in Kauf genommen, dass eine falsche Handlung gewählt und ausgeführt wird. Bei entsprechenden Anzeichen wird daher die momentan proaktiv ausgeführte Handlung abgebrochen und aufgrund der geänderten Situation eine neue Aktionswahl getroffen. Der Input des Moduls, die Wahrscheinlichkeitsdichteverteilung über die menschlichen Intentionen, wird zunächst gefiltert, um deren Verlauf zu glätten. Daraufhin wird die Menge Γ der möglichen Intentionen aus allen N Intentionen bestimmt, indem diese nach ihren Wahrscheinlichkeiten sortiert werden und die Position mit der größten Differenz zwischen diesen ermittelt wird. Die möglichen Roboteraktionen für die einzelnen Intentionen werden durch den Systemdesigner festgelegt. Ihnen werden Aktionstendenzen zugewiesen, und zwar abhängig von der Mächtigkeit von Γ auf unterschiedliche Weise. Im eindeutigen Fall ($|\Gamma| = 1$) oder unentscheidbaren Fall ($|\Gamma| = 0$ bzw. $|\Gamma| > 3$) ist die Vorgehensweise klar. Der interessante Fall liegt bei Mehrdeutigkeit und $1 < |\Gamma| \leq 3$ vor. Anstatt auf eine Aktion zu verzichten, wird hier eine Aktion proaktiv ausgeführt. Ziel ist es, die Unsicherheit der Intentionsschätzung zu minimieren und dem Benutzer ein explizites verbales Eingreifen zu ersparen. Im proaktiven Fall beeinflussen vier Kriterien die Aktionstendenzen: Die bedingte Entropie $H(I|A)$ als Maß für die Unsicherheit im System, der erwartete Nutzen einer Aktion $E \times V$, die nötigen Sicherheitsanforderungen, sowie die bisher häufigsten Aktionssequenzen. Die Aktionstendenzen werden durch ein Aktionsauswahlfilter in jedem Zyklus akkumuliert. Schließlich wird die Aktion ausgewählt, die die maximale akkumulierte Aktionstendenz besitzt.

Abstract

This thesis results from research on human-robot cooperation within the context of the collaborative research center “Humanoid robots—learning and cooperating multimodal robots”. It presents a haptic interface called “tactile language” that was designed to provide an additional non-verbal interaction modality. Furthermore, a method for proactive planning and execution of robot tasks based on the estimated human intention was devised. In order to integrate both methods into the existing simple robot control, an adequate robot architecture was conceived and implemented that comprises the existing robot control augmented by the additionally required architectural components.

The main structure of the robot architecture consists of three layers, where the upper two layers consisting of task planner and execution supervisor represent the main new contributions. The job of the task planner is to select the task that should be executed next. It was designed such that the robot can receive commands from all man-machine interfaces on the one hand, and from the proactive execution module on the other hand. To that end, an algorithm was defined to select the appropriate command in any situation. After the selection of a command, the execution supervisor ensures the correct execution of the task.

The tactile language developed in the course of this research allows a human to control the robot via its artificial skin or a touchpad. The processing system of the tactile language can be divided in three sections: First, it comprises the sensor interfaces and the tactile image processing; second, the recognition of the input symbols and characters of the language; and third, the interpretation of the input and the connection to the robot control. The tactile language offers the user four different input modes to choose from: The first two are a direct and an indirect control mode to move and rotate the robot’s TCP and head. The third is an abstract mode where symbolic, complex robot commands can be issued by entering appropriate multi-finger symbols. Furthermore, by integration of a freely available character recognition software, the user can enter handwritten alphabet characters. Lastly, there is a button mode that works like a touchscreen with active regions on the tactile surface that directly issue robot commands. The tactile language is very expressive since multiple parameters like direction, distance, and speed can be decoded from a single human finger stroke which can be used as attributes to the input symbol. Moreover, it is approximately real-time capable, thus making it possible to control the robot directly and recognize symbols instantly as well. The interpretation is done by a deterministic finite automaton that defines the tactile language and checks whether or not a tactile input makes sense. In case of a valid input the data is transmitted to the robot control.

A traditional service robot is controlled exclusively by explicit commands from the human. The paradigm of “proactive execution” that is newly established in this thesis contrasts this with an implicit command mode that aims at facilitating a more intuitive human-robot cooperation. Based on the estimated intention of the human cooperation partner the robot selects an appropriate action and starts to execute it unless the human has given an explicit command. In doing so, the robot risks selecting and executing the wrong action. Should this become evident, the robot thus aborts the currently proactively executed action and chooses another task based on the changed situation. The input of the module, the probability density distribution over the human intentions, is first filtered to smooth it. Then the set Γ of possible intentions of all N intentions is determined by sorting them according to their probabilities and identifying the largest difference between them. The appropriate actions for the individual intentions are predefined by the system designer. They are assigned action tendencies depending on the cardinality of Γ. In the unambiguous case ($|\Gamma| = 1$) or the undecidable case ($|\Gamma| = 0$ or $|\Gamma| > 3$) the procedure is clear. The interesting case of ambiguity occurs for $1 < |\Gamma| \leq 3$. Instead of idling, an action is executed proactively. The goal is to minimize the uncertainty of the intention estimation and spare the user an explicit verbal intervention. In the proactive case, four criteria influence the action tendencies: The conditional entropy $H(I|A)$ as a measure of the uncertainty of the system, the expected utility $E \times V$ of an action, the necessary safety requirements, and the most frequent action sequences so far. The action tendencies are accumulated each cycle in an action selection filter. Finally, the action is selected that exhibits the maximum accumulated action tendency.

Acknowledgment

I would like to thank my doctoral advisor, Prof. Dr. Heinz Wörn, head of the Institute for Process Control and Robotics (IPR) at the Universität Karlsruhe (TH). Without his continuous support, trust, and guidance, my research and this dissertation would not have happened. Additional thanks go to my co-advisor, Prof. Dr. Uwe D. Hanebeck, and his research assistant, Oliver Schrempf, for the good collaboration and helpful insights.

I have enjoyed working in the human-machine interaction group at the IPR with my fellow members PD Dr.-Ing. Catherina Burghart, Dirk Göger, and Nicolas Gorges. We had many fruitful discussions, joint projects, and great fun. The previous group members Oliver Kerpa, Dirk Osswald, and Dr.-Ing. Sadi Yigit laid the foundations which my research is built upon. I am also grateful for the other members of the IPR for providing a friendly and teamwork-oriented working environment.

This research was funded in part by the Collaborative Research Center SFB588 on "Humanoid robots—learning and cooperating multimodal robots" of the German Research Foundation (DFG), and I have appreciated the cooperation and help of my fellow researchers, especially Kai Nickel, Pedram Azad, Dr. Petra Wolf née Gieselmann, and Hartwig Holzapfel which provided components to integrate into our local demonstrator at the IPR.

Finally, I would like to thank my family: My wife Maria and my daughters Anna, Sophia, and Christina for their patience, support and putting up with frequent long or irregular working hours; my parents and my extended family for their support and encouragement.

Karlsruhe, July 2008 *Andreas Schmid*

Contents

1 Introduction

After a long dominance of industrial robots for manufacturing purposes, the world of robotics has been expanded to include service robots such as robotic lawn mowers or vacuum cleaners. More recently, a number of *humanoid robots* have been developed, such as the *ASIMO* that was created by the *Honda Motor Company* (see Fig. 1.1). When designing a humanoid robot, the goals are to imitate (a) the appearance, (b) the motor skills, (c) the perceptive faculty, and (d) the cognitive capabilities of a human being. This involves a broad range of scientific research areas from mechanical, electrical, and control engineering to computer science and artificial intelligence. For a specific robot design, there is usually a trade-off between the different goals depending on the scientific background of the development team.

This thesis results from research on human-robot cooperation within the context of the collaborative research center "Humanoid robots—learning and cooperating multimodal robots" [6]. The humanoid robot *ARMAR III* that arose from this collaborative research center is shown in Fig. 1.2. It was designed with a focus on human-like dexterity concerning manipulation tasks. As a consequence, it features lightweight arms and hands, plus a hip and a neck that all come close to human flexibility by their number of degrees of freedom. The head is equipped with visual and auditory sensors, and the body is partially clad in artificial skin for tactile sensing. The higher-level perception and the human-robot interfaces of the robot that are based on these sensors include a natural language dialog manager, a person and an object tracker.

As this research is addressing human-robot cooperation, its objectives are on improving the interaction between humans and the humanoid robot by making it more intuitive. As far as the intuitiveness is concerned, some progress has been achieved in the past by the development of natural language interaction systems, but there is still plenty to be done. Sharon Oviatt described this in her article "Ten Myths of Multimodal Interaction" [45]:

> ***Myth #2: Speech and pointing is the dominant multimodal integration pattern.***
> [. . .]
> ***Myth #4: Speech is the primary input mode in any multimodal system that includes it.*** [. . .] *In short, speech is neither the exclusive carrier of important content, nor does it have temporal precedence over other input modes. As a result, the belief that speech is primary risks underexploiting the valuable roles to be played by other modes in next-generation multimodal architectures.*
> ***Myth #5: Multimodal language does not differ linguistically from unimodal***

Figure 1.1: The humanoid robot *ASIMO* created by *Honda Motor Company*.

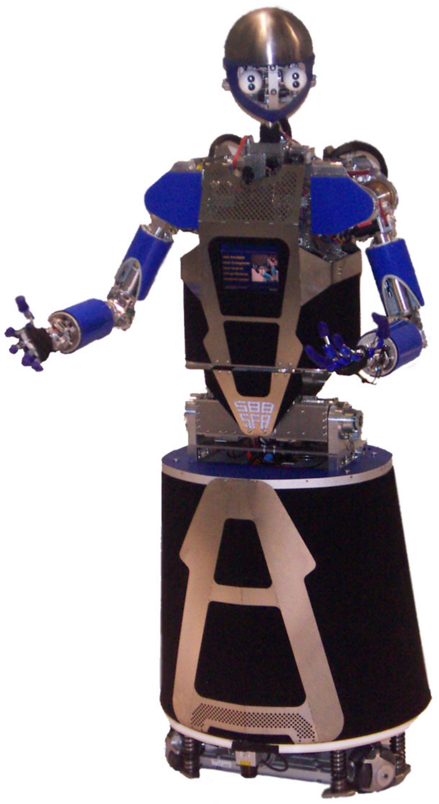


Figure 1.2: *ARMAR III*, the demonstrator robot of our collaborative research center.

> ***language.*** [...] *In fact, it recently has been demonstrated that multimodal pen/voice language is briefer, syntactically simpler, and less disfluent than users' unimodal speech.*

This thesis hopes to contribute to alleviate some of these issues. The objectives that were established in this sense are detailed in the following section.

1.1 Objectives

The goal of intuitive human-robot interaction was pursued by providing a new *explicit* way to command the robot using the tactile modality via the robot's artificial skin, as well as furnishing an *implicit* way of human-robot interaction drawing on the estimation of the human intention. In order to integrate these new human-robot interfaces into a conventional robot control, suitable architecture components had to be designed.

For the new tactile robot control mode, which we call "tactile language", the focus lay on the following objectives:

- Provide a redundant human-robot interaction modality next to natural language or gesture recognition.
- Offer new ways of robot control through leveraging the unique properties of the robot's artificial skin as a spatial tactile sensor.
- Design the interface to work independently from specific hardware, i.e., allow the use of arbitrary tactile sensors beyond our own artificial skin.
- Low computational demands to achieve approximately real-time performance.
- Intuitive usage for new, untrained users.
- Good extendability such that new commands can be easily added.

The design of the implicit robot command interface, the "proactive execution module", was guided by these goals:

- Enable intuitive human-robot cooperation in the sense of avoiding explicit clarification dialogs and explicit commands by reading the user's non-verbal cues.
- Select an appropriate robot action based on the estimate of the human intention.
- Start to execute a task even when uncertain about the true intention of the human, thus acting "proactively".

- Trigger clarifying reactions to get more information and reduce the uncertainty in the system. The challenge is to select a robot action that urges the user to react in a way that unravels the user's intention.

The addition of these two novel interfaces to the robot required the conception of new robot architecture layers adhering to the subsequent requirements:

- The integration of a variety of man-machine interfaces must be feasible to allow for multimodal interaction with the robot. Sometimes, it is more natural for a human to express a command verbally, and sometimes a gesture or a touch suffices.
- The interaction with the robot should be as intuitive as possible, especially for non-technical users. This demands not only a natural language system, but also the possibility for implicit communication through intention recognition.
- To be accepted in a human environment, the robot needs to act dependably, reliably, and predictably. Moreover, the human needs to be in control at all times; this requires to adopt the idea of keeping the human in the loop.

1.2 Outline

The remainder of this thesis is structured as follows: Chapter 2 examines the state of the art in the fields of robot architecture, tactile human-machine interaction, and proactive robot behavior. Chapter 3 presents the robot architecture additions that were required to accommodate the new explicit and implicit command interfaces. Next, chapter 4 describes the design and evaluation of the tactile language. Following that, the concept and functionality of proactive execution is detailed in chapter 5. Chapter 6 finally gives conclusions and an outlook.

2 State of the Art

In this chapter, the current state of the art in the areas of robot architecture, tactile man-machine-interaction, and proactive execution of tasks is examined.

2.1 Robot Architecture

The number of published robot architectures is vast. E. Gat wrote an extensive survey on *three-layer architectures* [17]. It reviews the evolution of robot architectures from the traditional *sense-plan-act* (SPA) style prevalent in architectures at the beginning of robotics until the mid-80s of the last century. It is characterized by a sensing element that processes raw sensor data to acquire a high level perception of the world, i.e., a world model, then a planning module that generates a plan of actions based on the current state of the world and one or a set of goals that the robot is to achieve, and finally an execution layer that executions the actions according to the plan. According to Gat, several shortcomings of the SPA style became all too obvious around 1985; they include symbolic planning and comprehensive world modeling that proved to be very hard problems, and the open-loop plan execution that was incapable of dealing with the uncertainty of real world problems. A new paradigm, the behavioral approach, was introduced by R. Brooks with the subsumption architecture [10] that shunned symbolic representations but instead is composed of simple finite state machine networks connected by wires, each implementing a *behavior*. A hierarchy of behaviors can be achieved in that the output wires of higher level networks suppress or inhibit the output of lower levels. Subsumption-based robots turned out to be really successful at collision-free navigation, but lacked the ability to achieve more complex tasks. Thus, the current trend of *hybrid architectures* that combine elements of both arose. Gat's survey determined that three-layer architectures consist of the following three layers: a *controller* that represents the lowest level of various feedback control loops driving the actuators in a real-time fashion, a *sequencer* that supervises and parameterizes the execution the various actions of the current plan one after the other, and a *deliberator* at the highest level that creates the plans for the sequencer to execute. They are distinguished by their use of internal state and their execution cycle times, ranging from no internal state and real-time operation at the bottom to complex world models and time-consuming exponential search algorithm executions on top. These Controller-Sequencer-Deliberator layers inspired the Task Coordination & Task Execution-Execution Supervisor-Task Planner layers presented later in this work.

E. Coste-Manière et al. [14] make the case for the importance of robot architectures that

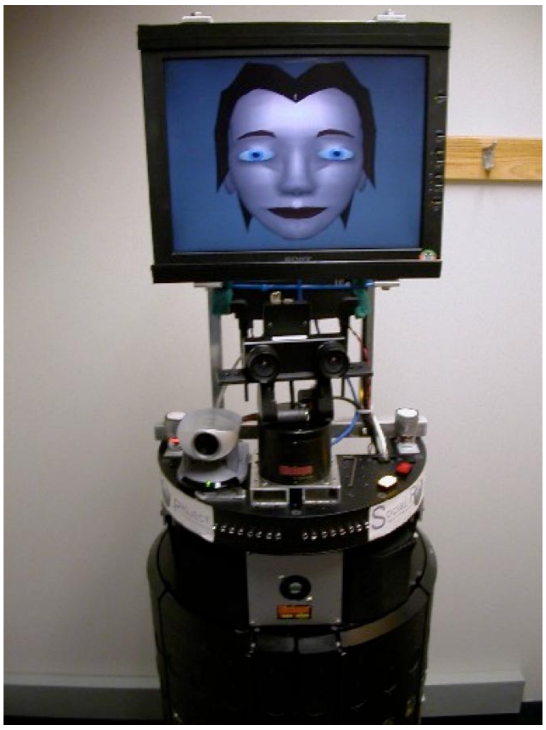


Figure 2.1: The robot GRACE [57].

adhere to a structural hierarchy and a style of computational concepts. It details design requirements for robot architectures such as the skills to achieve high-level complex goals, to handle uncertainty, to interact with a complex dynamic environment, to operate in real-time, and to consider system safety issues; it also presents corresponding architecture examples to each of those skills.

An interesting work regarding a complex robot system is the autonomous robot GRACE (Graduate Robot Attending ConferencE) [57] depicted in Fig. 2.1. It was built by R. Simmons et al. as a joint effort of several research institutions to take part in the 2002 AAAI Robot Challenge. The task was for the robot to attend the AAAI National Conference on Artificial Intelligence as a participant, thus requiring the subtasks of finding the registration booth and registering, interacting with people, using a map of the premises to find its way to the presentation venue in time, and giving a technical presentation there about itself. In order to achieve the task, GRACE incorporates a range of Man-Machine Interfaces (MMIs) with some of them similar to the ones used in our collaborative research center—a stereo camera for tracking and gesture recognition of humans, a close-range microphone for natural language communication, a single color camera with pan-tilt zoom capability, plus a SICK laser scanner, touch, infrared and sonar sensors integrated into the B21 platform. They are integrated by the software architecture shown in Fig. 2.2. Overall, it can be said that GRACE includes impressive state-of-the-art skills such as localization and mapping, natural language dialogues, and collision-free navigation, but that they all were specifically tailored to achieve the aforementioned subtasks in that

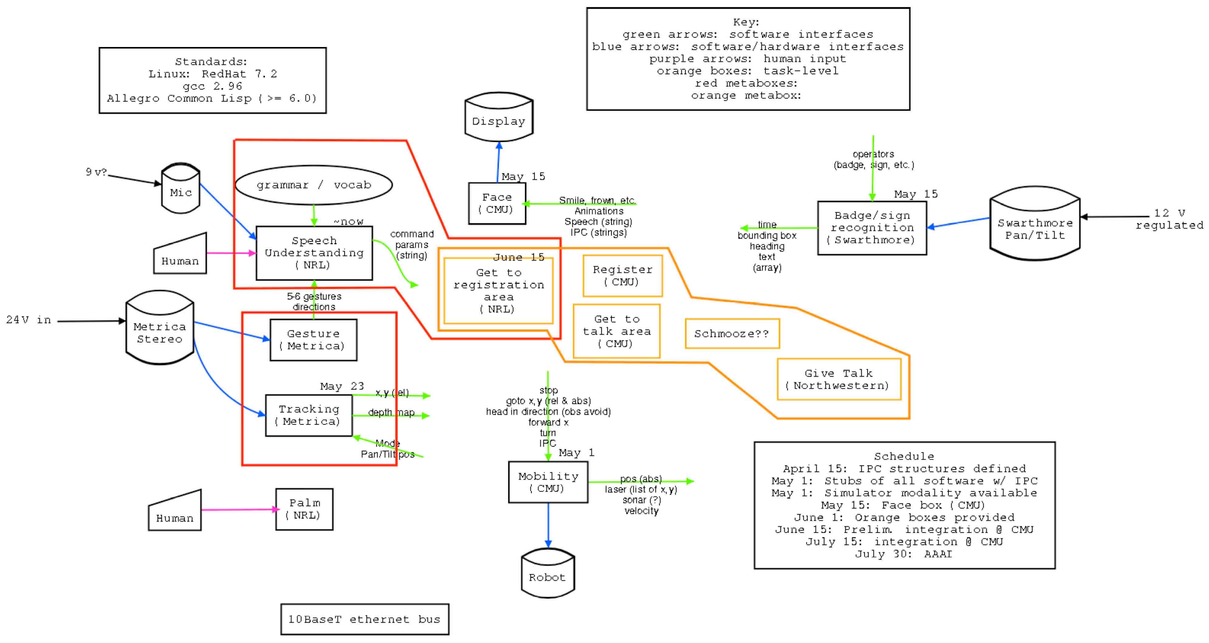


Figure 2.2: The GRACE software architecture [57].

particular conference center. Thus, the attribute "autonomous" is a somewhat bold statement, and the architecture represents more an *ad hoc* solution than a structured approach.

An "integrated system for cooperative man-machine interaction" was presented by a research group in Bielefeld [4]. At the time of its writing only the individual components were developed, and their integration into the proposed system architecture (cf. Fig. 2.3) was still to be done. The architecture focuses almost exclusively on MMIs, with a demonstration scenario of toy building blocks and a human instructor that issues commands on how to assemble the individual parts together, which the system has to understand by fusing natural language commands and computer vision. A related project with a similar focus is the Bielefeld Robot Companion (BIRON) [21]. This robot system is comprised of a mobile platform carrying a screen, a laser range finder, a camera, and stereo microphones. The main components of the BIRON architecture displayed in Fig. 2.4 are the person attention system, speech recognition and understanding, and the dialog manager. It should be noted that only the components in solid line boxes had been implemented thus far. Both of the aforementioned systems are lacking actuators beyond a platform, such as manipulators or grippers with which to physically interact with the environment. Thus, the range of possible physical actions is either non-existent or limited to platform navigation, which results in considerably simpler requirements to their architectures. Specifically, the set of actions that have to be planned and sequenced is rather limited, as is the possibility of hardware failures and exceptions. In contrast, the architecture presented in this work in chapter 3 is targeted towards a multi-purpose robot whose MMIs serve to trigger robot actions which create the actual value for the user, not the MMIs themselves.

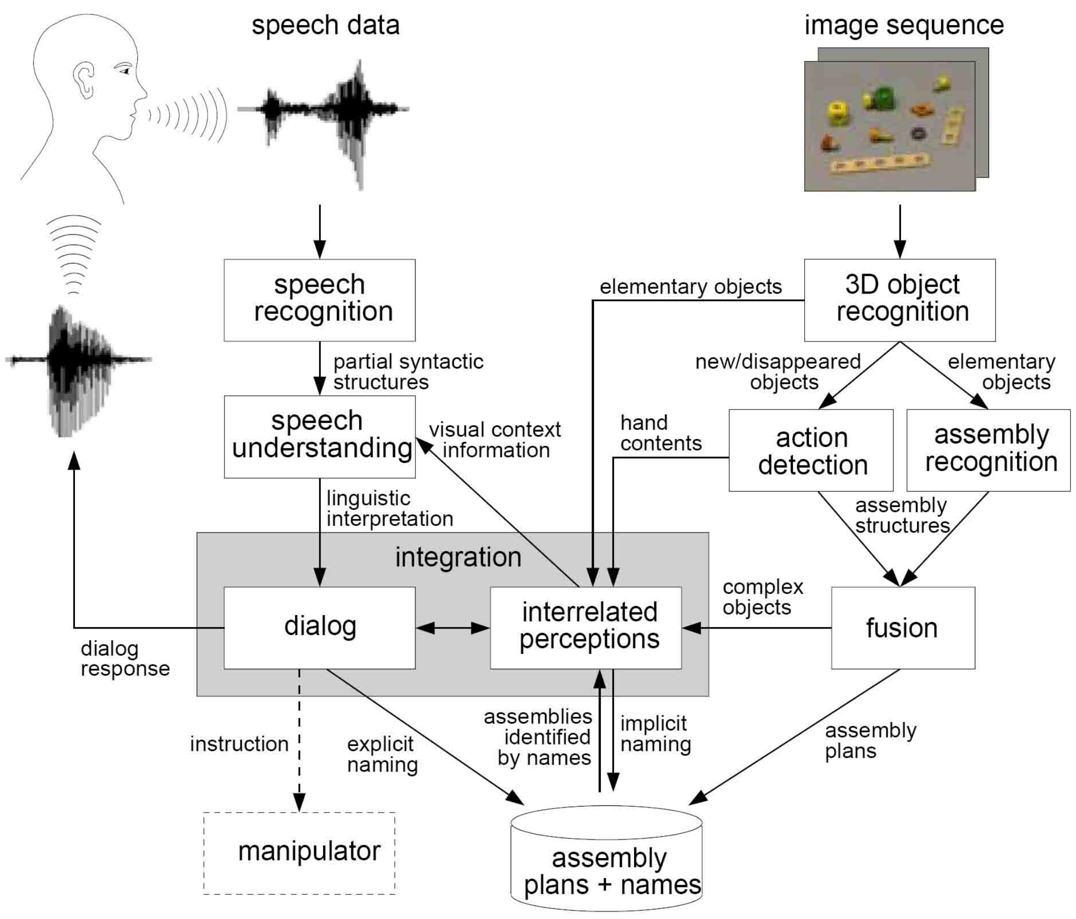


Figure 2.3: Proposed system architecture from [4].

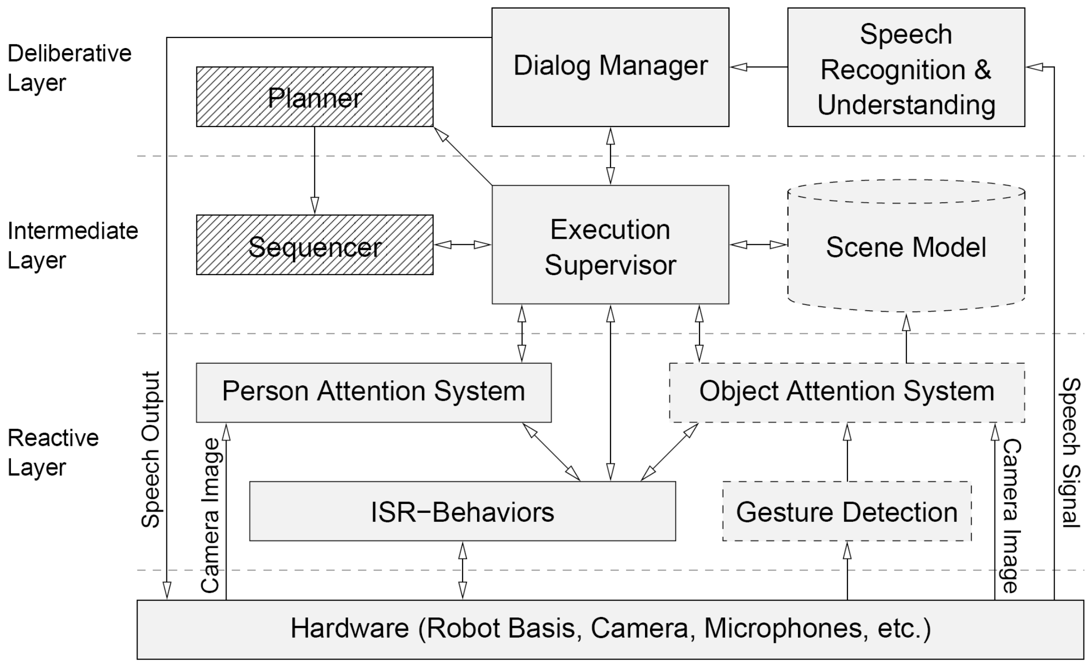


Figure 2.4: The BIRON architecture [21].

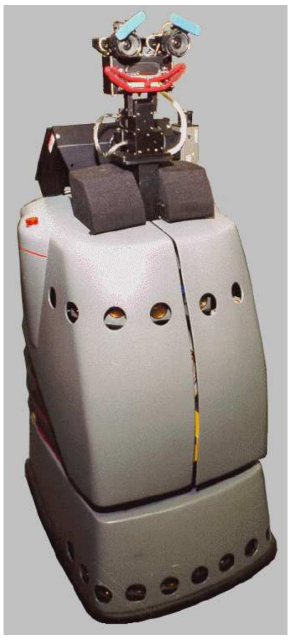


Figure 2.5: The tour guide robot Minerva [63].

MINERVA [63] represents a robot that operated as a museum tour guide robot in a human-populated environment, see Fig. 2.5. Its main research components are localization and mapping, collision avoidance, path planning, spontaneous short-term interaction, and a Web interface, and their design follows the three main principles of probabilistic reasoning, learning, and the use of any-time algorithms. Despite the robot's richness in features and functionality, its actual software architecture remains largely undescribed, however. It is mentioned that it consists of 20 distributed modules that communicate asynchronously using an inter-process communication system named TCX, and that the modules belong to 4 different hierarchy levels starting from a hardware interface layer over a navigation and an interaction layer to the Web interface layer. Beyond that, no information is given on the identities of the modules and their collaboration, thus it can be suspected that they were stitched together in an *ad hoc* fashion similar to GRACE's approach to solve the particular tour guide robot tasks, namely guiding visitors while navigating through the exhibits.

Another example of a service robot is the autonomous service robot PSR (Public Service Robot), whose job is to perform transportation and surveillance tasks in public buildings such as offices or hospitals [34]. Its architecture is shown in Fig. 2.6. It follows the hybrid architecture paradigm as on the one hand, it is divided in three layers—deliberate, sequencing, and reactive layer (from top to bottom); on the other hand, the reactive layer has a behaviors component that that executes behaviors which generate basic motions using tight sensor-actor control loops. As with all the other robots in this section, PSR acts only when explicitly commanded to do so through the human-robot interface (HRI) where users can input numbers. The commands consist of two room numbers, the initial and the target room number that specify the parameters

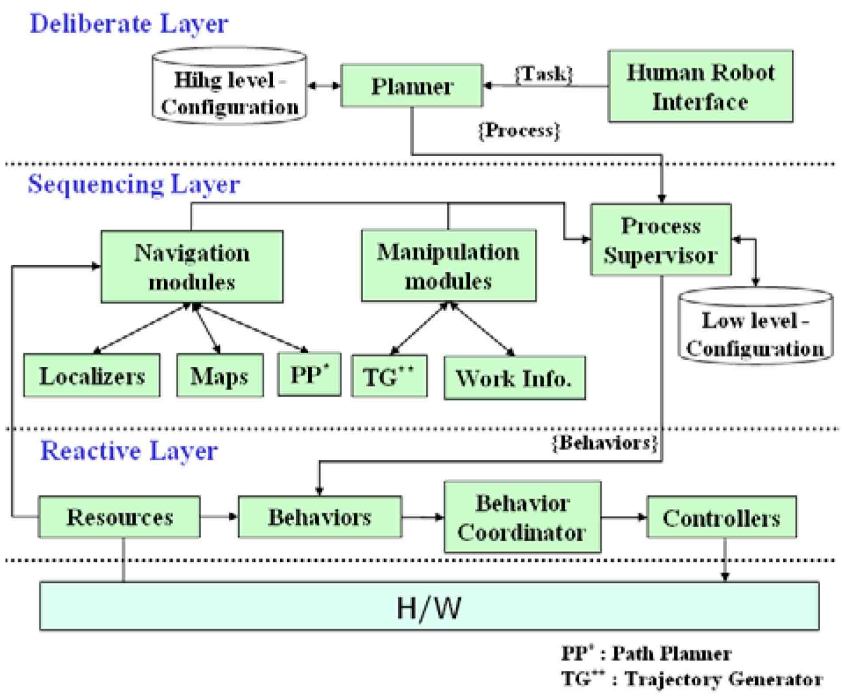


Figure 2.6: The Public Service Robot PSR [34].

for the document transportation task which seems to be the only task of PSR at the moment. The task is supervised and executed by both a high-level and a low-level Petri net which is said to be generated automatically, though it does not come clear how exactly this is done. It seems most likely that the Petri net structure itself is fixed, and only the parameters of initial and target room locations change. Although the structured architecture is emphasized, the main focus of PSR seems to be safe collision-free navigation through office spaces, and the MMI is limited to a minimum.

Finally, a well-known control architecture called TCA (Task Control Architecture) was developed and extended by R. Simmons [58], which was, for example, employed for the autonomous mobile robot Xavier [56]. An example implementation for the Ambler walking system is displayed in Fig. 2.7. Despite its name, TCA is less an architecture than rather a generic control framework for coordinating distributed modules. It is comprised of asynchronous task modules that implement the robot's functionality, and a central control module that plans, executes, and synchronizes tasks through a message passing communication system. Specifically, the central control module supports distributed inter-process communication, task decomposition and temporal constraints between subtasks, resource allocation and management, execution monitoring, and exception handling. This support consists of providing different kinds of messages: Inform, query, goal, command, monitor, and exception messages. It is up to the specific robot control implementation to provide the modules that generate and process these messages appropriately, while TCA merely facilitates the distribution of those messages. The major challenge for TCA

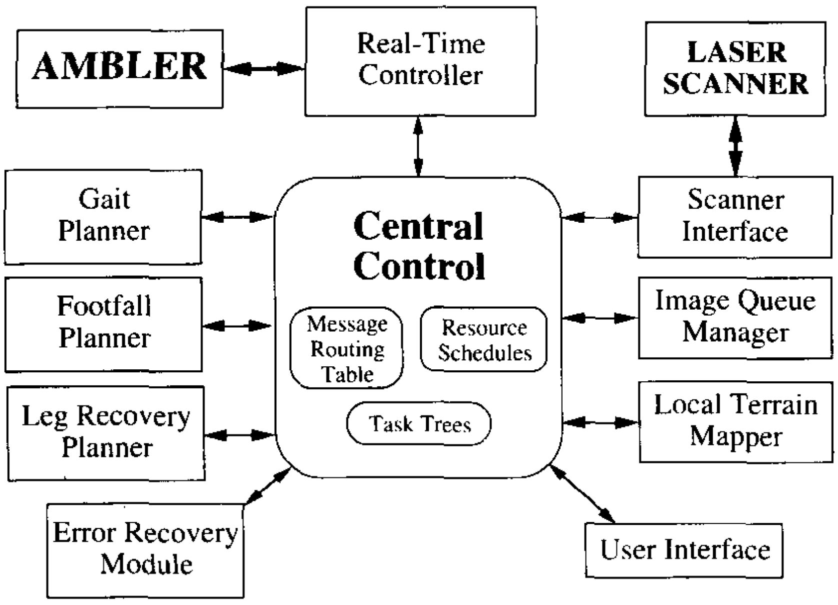


Figure 2.7: Modules for the Amber walking system using the Task Control Architecture TCA [58].

is preventing the central control module from becoming a bottleneck for distributed systems with many modules, since every message is routed through it.

In summary, a lot of robot architectures have been published so far. However, these publications often times do not go into details on how the architectures actually work, but rather stay on the surface by providing high-level block diagrams; in addition, most are not available as open source code either, thus preventing other interested research groups from using them. Also, these architectures often represent custom solutions tailored to specific tasks and hardware that makes it difficult to speak of actual structured "architectures". Most importantly for this work, true robot autonomy is seldom supported; instead, users are required to explicitly command the robot through the robots' respective MMIs.

2.2 Tactile Language

Early tactile languages were developed for human-human communication, specifically to enable deaf people to make themselves understood. They are also subsumed under the notions "fingerspelling" or "dactylology". One example is the manual alphabet by Dalgarno (Fig. 2.8) that he published in 1680 [15]. It works by assigning the different letters of the alphabet to the finger joints of the left human hand. By pointing to the respective joints, one is able to spell words and sentences. Another example is the Lorm-alphabet created in 1881 by Hieronymus Lorm who was both deaf and blind in his later life. Again, the letters of the alphabet can be designated by touching different parts of a human hand. Even though they might be appreciated by people with disabilities that have to rely on the tactile modality, these manual languages are

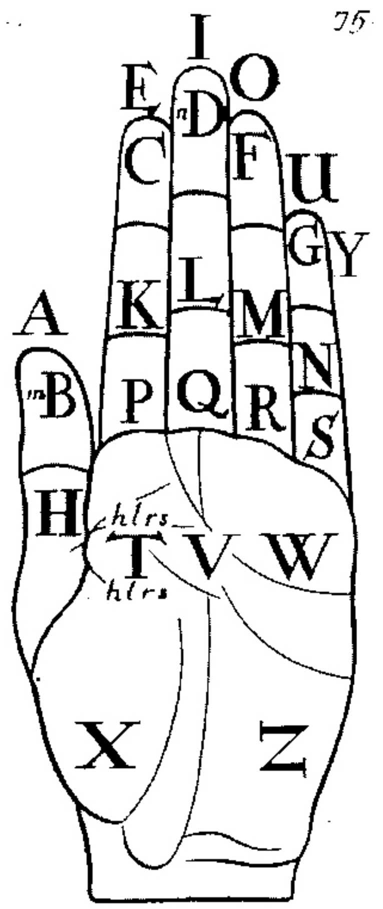
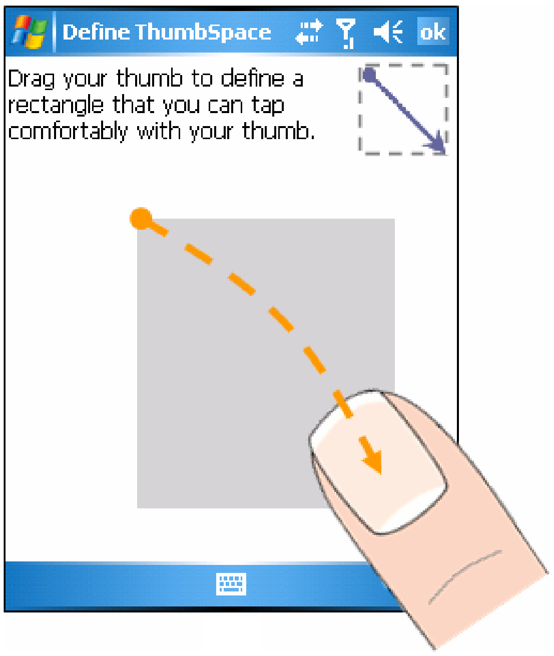


Figure 2.8: Manual alphabet by George Dalgarno [15].

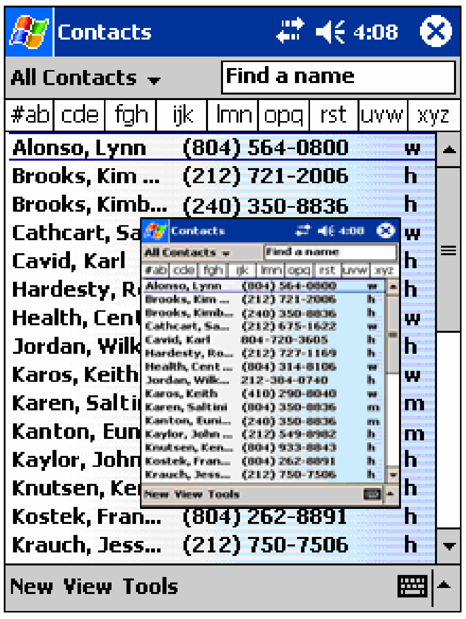
very restricted in their number of users. In our times, tactile communication is usually employed to interact with machines and electronic devices.

One example is the touchscreen interface for mobile devices that Karlson and Bederson presented. It is operated one-handedly by using the touchscreen with the thumb of the hand that is holding the device [32, 31]. By moving the thumb, the user can move a cursor and select objects. Since the touchscreen is usually larger than what can be comfortably reached with the thumb, the user needs to define a subspace of the touchscreen called “ThumbSpace” that he wants to use (Fig. 2.9(a)). The actual display is then linearly mapped onto this virtual subspace for all tactile interaction purposes such as navigation or selection (Fig. 2.9(b)). Initial user tests showed that it was quite difficult to select entities on the screen that are relatively small or even tiny compared to a thumb. Thus the authors eventually came up with a three-step selection method: First, the ThumbSpace acts as an absolute touchscreen with a linear mapping compared to the actual screen, and the user can make an initial try to hit the desired screen item (Fig. 2.9(c)). Then, the ThumbSpace turns transparent white and transforms itself into a relative selection space—by moving the thumb, the selection cursor moves to the adjacent objects (Fig. 2.9(d)). Finally, the user confirms his selection by lifting the thumb up (Fig. 2.9(e)). In a



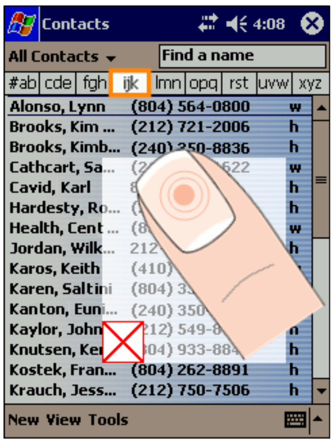
Drag your thumb to define a rectangle that you can tap comfortably with your thumb.

(a) Defining ThumbSpace.

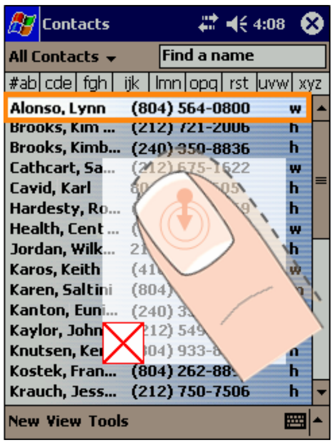


Alonso, Lynn	(804) 564-0800	w
Brooks, Kim ...	(212) 721-2006	h
Brooks, Kimb...	(240) 350-8836	h
Cathcart, Sa...	(212) 675-1622	w
Cavid, Karl	804-720-3605	h
Hardesty, Ro...	(212) 727-1169	h
Health, Cent ...	(804) 314-8106	w
Jordan, Wilk...	212-384-0740	h
Karos, Keith	(410) 290-8040	w
Karen, Saltini	(804) 350-8836	m
Kanton, Euni...	(240) 350-8836	m
Kaylor, John ...	(212) 549-8982	h
Knutsen, Ken...	(804) 933-8843	h
Kostek, Fran...	(804) 262-8891	h
Krauch, Jess...	(212) 750-7506	h

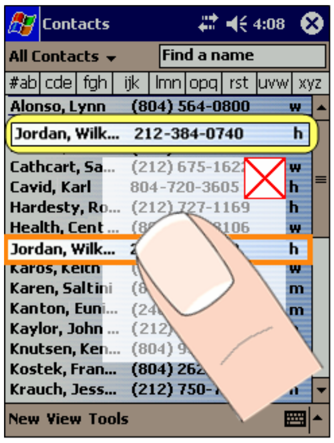
(b) ThumbSpace as a Radar View.



(c) Selection – Aiming.



(d) Selection – Adjusting.


Contacts
4:08
All Contacts
Find a name
#ab cde fgh ijk lmn opq rst uvw xyz
Jordan, Wilk... 212-384-0740 h
New View Tools

(e) Selection – Confirmation.

Figure 2.9: ThumbSpace—operating the touchscreen of a mobile device with one's thumb [32].

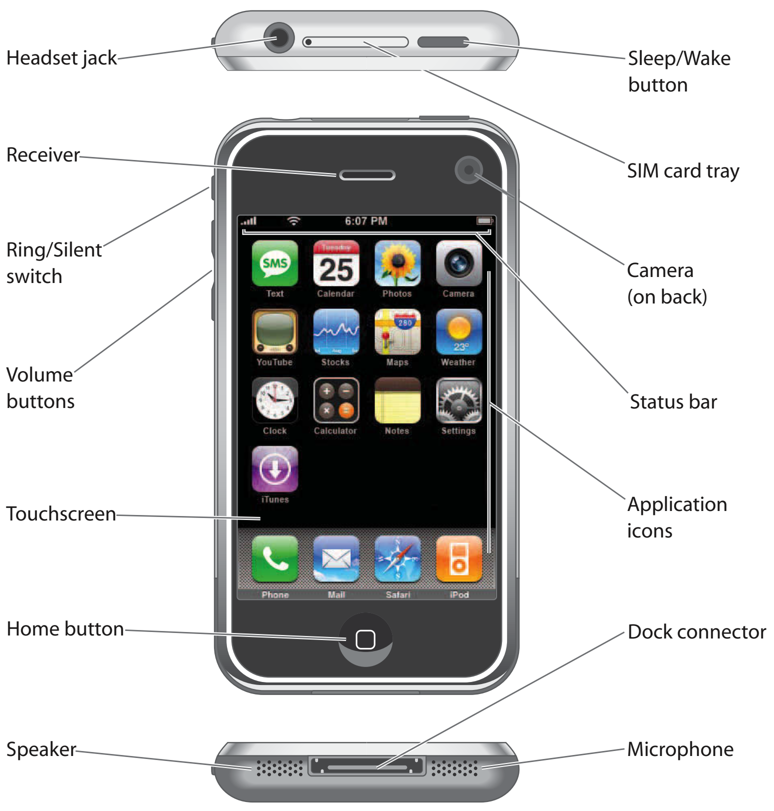


Figure 2.10: The *Apple iPhone*^{TM} [1].

subsequent user study, ThumbSpace was tested against other selection methods such as direct touch (i.e., trying to hit the desired item directly on the actual screen instead of using ThumbSpace) and selection with peripheral input hardware (e.g., a scroll wheel). It turned out that the ThumbSpace method only yielded mediocre results in both accuracy and user satisfaction, with the main disadvantages being the occlusion of a part of the display by the ThumbSpace window, and the lack of accuracy when trying to select small objects with the relatively big thumb. Our tactile language does not suffer from these issues, however, since the single-finger input can be performed with any finger the user is comfortable with, and the multi-finger input does not require the precise handling of small items. Furthermore, due to our different system design, occlusion is not a problem, either.

A lot of hype has been created by the *Apple iPhone*^{TM} [1]. Apart from a single *Home* button, its main user interface consists of a multi-touch screen that is based on capacitive sensing technology (see Fig. 2.10). For the sensor data processing, *Apple Inc.* hired the main developers of the company *Fingerworks* that had developed and produced the *iGesture Pad* (cf. Fig. 4.3 on p. 45) that we have been using for our tactile language, and also acquired their patent; since then, *Fingerworks* has ceased to exist. Thus, the *iPhone* can be viewed as a successor of the *iGesture Pad* with similar hardware capabilities, apart from the different size. As such, the *iPhone* allows the user to select icons on the screen by touching them, and flip through lists by

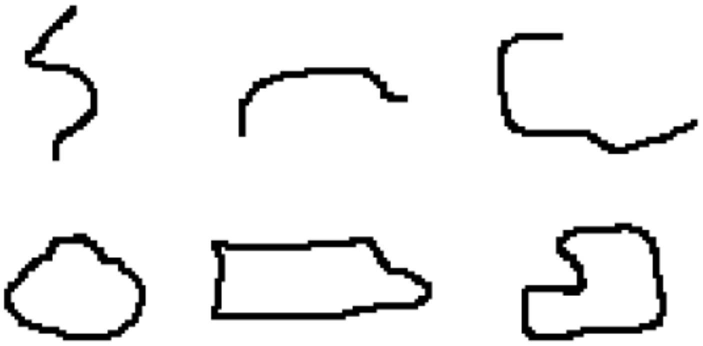
Figure 2.11: The *Microsoft Surface*^{TM} [39].

dragging the finger across the display, with neither action requiring a contact force. The multi-touch capability comes into play, e.g., when looking at photos—they can be zoomed in and out by spreading apart or closing two fingers. For text and number input when using the calculator or typing a phone number or an SMS, a virtual keyboard or number pad are displayed where the user can type on the keys directly. Since the hit accuracy is usually limited, an automatic spell checker corrects the typed words online. In comparison to our own tactile language, the button or touchscreen mode and some of the symbols and dragging motions are basically equivalent. Also, in principle the *iPhone* could be used as a tactile input sensor in the same way as the *iGesture Pad*. The difference lies in the applications, i.e., controlling a real-world robot versus operating a graphical user interface, and the limited tactile interaction area of the *iPhone*.

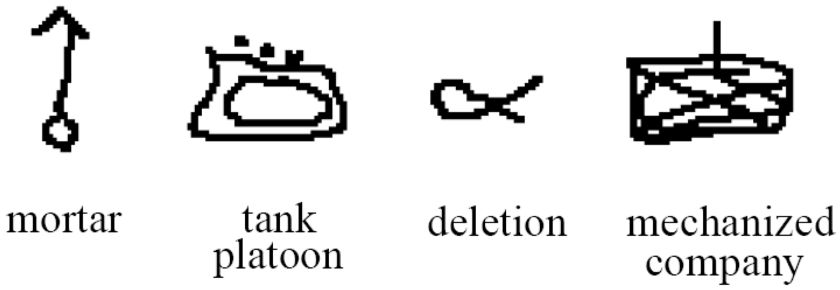
Somewhat related to the *iPhone*, though on a larger scale, is *Microsoft*'s *Surface*^{TM} [40, 39]. It is a table that consists of a black base and a translucent surface (cf. Fig. 2.11). It features a computer in its base and a 30 in touchscreen on top where users can tactually interact with the display by touching, dragging, and manipulating objects on it, also with multiple fingers at the same time. Thus, as far as the tactile interaction is concerned, *Surface* compares to our tactile man-machine interface much like the *iPhone*.

The "Interaction Complementarity Assignment Redundancy Equivalence" (ICARE) graphical platform [8] is used to design multimodal interfaces. One application that has been developed with it is a military aircraft simulator named "INTUITION Avionic Simulator" (IAS) whose interface includes the haptic components of two command joysticks for piloting and targeting and a tactile surface between the legs for switching equipment on (cf. Fig. 2.12). IAS also features speech command input. For critical tasks (e.g., switching from navigation mode to

Figure 2.12: The IAS real-time flight simulator [8].



(a) Pen drawings of routes and areas.



(b) Pen input of map symbols and editing gestures.

Figure 2.13: Pen-based gesture input from the QuickSet system [13].

fighting mode), users are required to issue the respective command via two modalities. The tactile surface displays five graphical buttons that correspond to five pieces of equipment such as Radar or Head Up Display. This corresponds to the touchscreen mode of our tactile language; there is no other form of input possible, however.

The multimodal system QuickSet [13] is based on a multiagent architecture. The user interface consists of a handheld PC that features a speech recognition agent and a pen-based

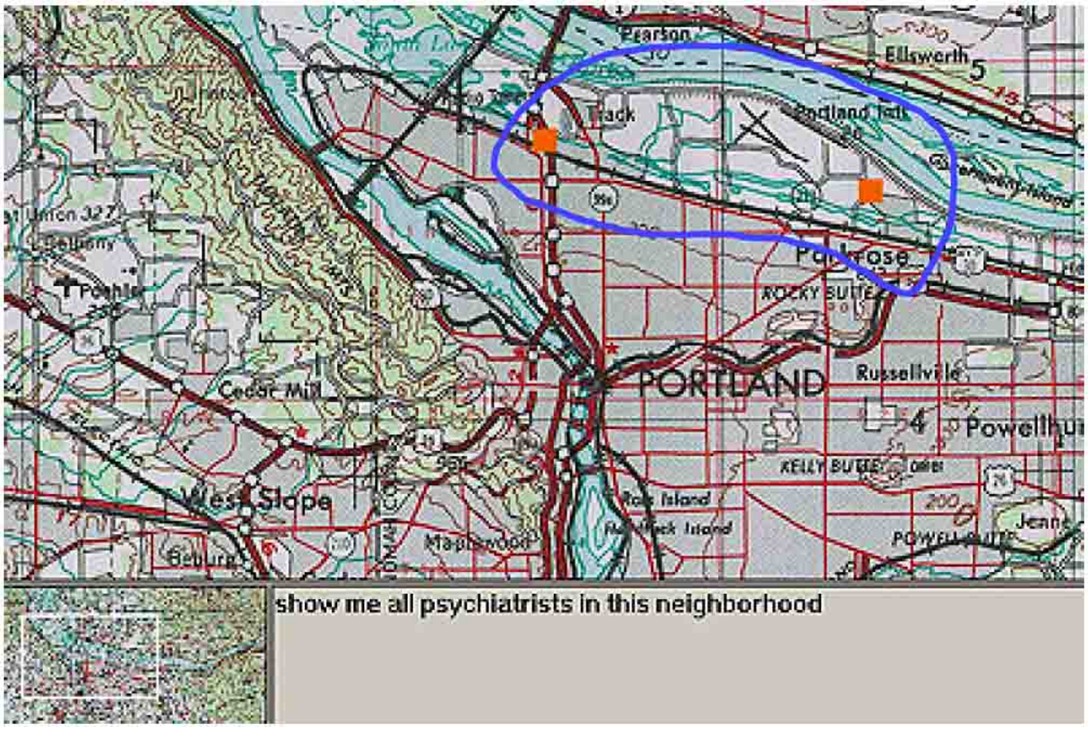
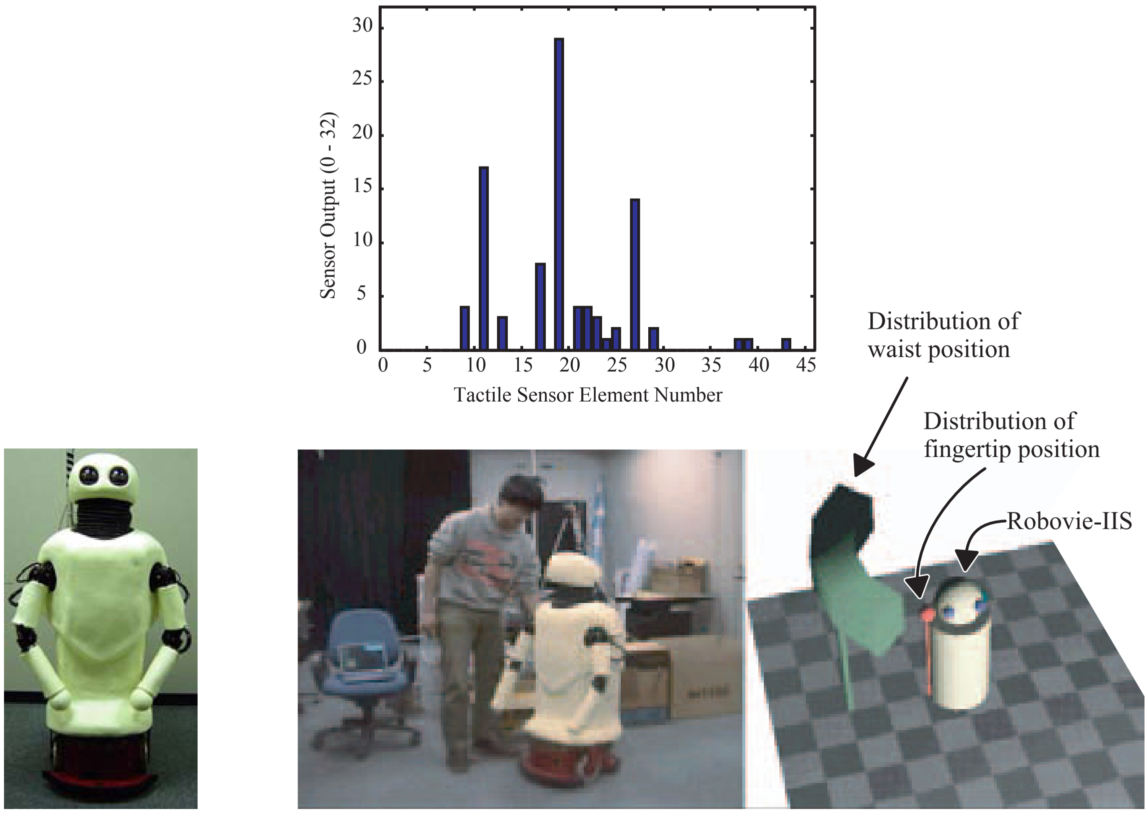

show me all psychiatrists in this neighborhood

Figure 2.14: QuickSet-based application “Multimodal Interaction with Medical Information” (MIMI) [13].

gesture recognition agent; its screen is used to display maps of varying resolution. A user can enter voice commands and mark areas (Fig. 2.13(a)) or draw objects and editing gestures (Fig. 2.13(b)) on the map with the pen as command parameters; both speech and tactile input form a combined command. The gesture recognition is able to recognize 68 different pen-gestures such as military map symbols (tank, mortar, platoon, etc.), editing gestures (deletion, grouping), routes, and areas. The recognition is done in parallel with both a neural network and a set of hidden Markov models (HMMs), where the estimations of both recognizers are combined to yield combined probabilities for all possible interpretations. In order to improve suboptimal recognition results due to sloppy input, the gesture recognition output is fused with the speech recognition output into typed feature structures that represent the utterance meaning. An example of an application of the QuickSet system is shown in Fig. 2.14. The application is called “Multimodal Interaction with Medical Information” (MIMI) and allows users to make inquiries about available health care providers in a certain area using speech and gesture input. The example query in the figure is “show me all psychiatrists in this neighborhood <circling gesture on the map>”. According to this input, the system queries a database of doctor records and displays the results as icons on the map. In their conclusion, the authors remark that the system benefitted from the multimodal input, although it became apparent that in real-world situations subjected to considerable noise the speech input was not usable; thus, a complete functionality overlap between speech and tactile input was envisioned as a future improvement.


Sensor Output (0 - 32)
Tactile Sensor Element Number
0
5
10
15
20
25
30
35
40
45
Distribution of waist position
Distribution of fingertip position
Robovie-IIS

(a) Robovie-IIS. (b) Subject touches right side of Robovie-IIS.

Figure 2.15: Haptic communication with Robovie-IIS [42].

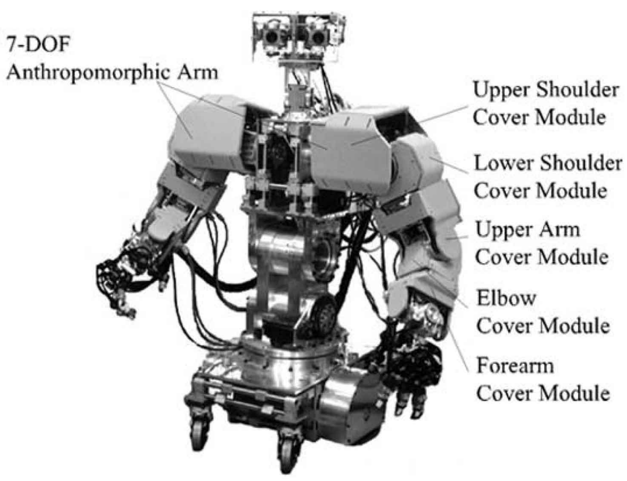
Compared to our tactile language, QuickSet provides functionality that is comparable to our symbol and indirect input mode. However, it lacks modes that allow the user to directly control the movement of an object, nor does it possess a touchscreen mode for a quick selection of commands by touching predefined screen areas.

As to strict human-robot communication, only a few examples can be found of related work using tactile or haptic information apart from low-level guiding of a robot arm by zero-force control [19]. A very rudimentary symbolic tactile interaction mechanism is described in [70] where the tactile skin of a robot gripper is touched back and forth left-right-left-right within one second to signify the gripper to close itself. A similar motion is then used to make the gripper open and release the grasped object.

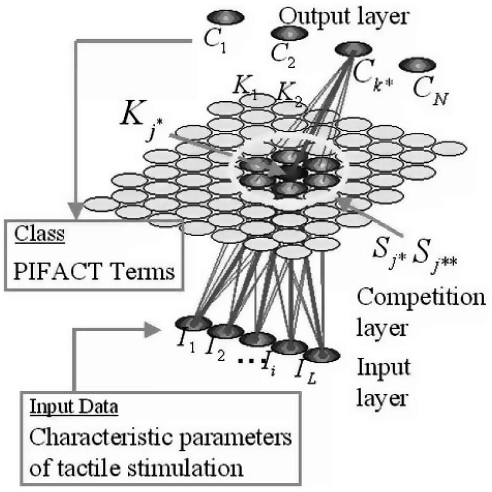
Miyashita et al. describe what they call "haptic communication" [42]. This communication takes place between a human and their robot Robovie-IIS shown in Fig. 2.15(a), which is a version of the Robovie series whose body is covered by 48 distributed film-type polyvinylidene fluoride (PVDF) sensors that output a high voltage proportionate to changes in applied pressure. The system estimates waist and hand positions of the human based on the tactile sensor input, which is deemed useful for encounters between the robot and humans where the camera pictures cannot capture the whole scene anymore due to the humans being too close to the

touch	beat	pick	scrub
hit	push	pull	grasp
collide	thrust	tug	grip
smite	poke	draw	seize
pat	jab	drag	pinch
tap	jog	tweak	stroke
slap	nudge	scrape	scratch
punch	prod	rub	

Table 2.1: PIFACT terms [29].



(a) The robot WENDY.



(b) The MCP neural network.

Figure 2.16: Human-robot contact classification [29].

robot. To that end, the authors recorded ground truth data with a VICON motion capture system and the associated tactile sensing data for some selected human postures around the robot. After filtering out irrelevant touch data, the remaining tactile and visual data samples were each clustered. Finally a mapping between visual and tactile data clusters was done. To use this mapping, the current tactile sensor vector is obtained and the closest tactile cluster calculated. Then the corresponding visual clusters are determined from which probability distributions for the waist and hand positions are computed. An experiment of a subject touching the right side of Robovie-IIS is displayed in 2.15(b).

Iwata and Sugano proposed a method to classify human-robot contacts that are detected by the robot's artificial skin [29]. The types of contacts they tried to classify cover the entire range from accidental to deliberate contacts, both active and passive. They summarize them as Physical InterFerence and intended contACT (PIFACT) with humans. Table 2.1 lists the English terms they used to label the different contact classes. For learning and experimenting, the authors relied on the robot WENDY (Fig. 2.16(a)). It possesses two 7-DOF anthropomorphic arms that are equipped with several force sensing resistors (FSRs) for measuring the contact forces. As classification method, Iwata and Sugano chose a self-organization-based neural network called modified counterpropagation (MCP) for reasons of simplicity. It is depicted in Fig. 2.16(b). It consists of L input neurons I_i for the L-dimensional PIFACT recognition parameter vector. They used between 6 and 11 input parameters such as maximum force, sum of contact area, or maximum two-point distance. The output layer has $N = 10$ neurons C_k that correspond to the subset of PIFACT terms that are in the gray shaded table cells in Table 2.1. The competition layer is composed of an array of $20 \cdot 20 = 400$ neurons K_j. Each of these neurons is connected to each input layer neuron by a weighted connection w_{ji}, thus forming the connection-weight vectors $\underline{W}_j = [w_{j1}, w_{j2}, \dots w_{jL}] \in \mathbb{R}^L$. Likewise, the neurons are also connected to the output layer by the connection-weight vector $\underline{Z}_j = [z_{j1}, z_{j2}, \dots z_{jN}] \in \mathbb{R}^N$. In order to train the network they collected sample data by first performing a human-human contact with the recipient labeling the type of contact, then applying the same contact type to WENDY by the human contact initiator. Subsequently, each recorded input parameter vector $\underline{X}(t_k)$ was compared to all vectors $\underline{W}_j$ to determine the winning neuron K_{j^*} whose weight vector $\underline{W}_{j^*}$ was of minimal Euclidean distance to $\underline{X}(t_k)$. Then $\underline{W}_{j^*}$ and the weight vectors of a neighborhood S_j were adjusted in direction of $\underline{X}(t_k)$ to gradually cluster the weight vectors. A similar process was applied to train the vectors $\underline{Z}_j$. After this learning phase, new input vectors could be identified via the winning neuron K_{j^*} and its output connection vector $\underline{Z}_{j^*}$ that yields a probability distribution over all N contact types after a normalization step. The experimental results with a specially designed tactile shoulder sensor show that this contact classification method works in principle. There is no concept, however, as to how these classified contacts should be acted upon. Thus it is only a tactile perception method, but no actual robot command interface.

2.3 Proactive Execution

Most applications of proactive behavior are located in the realm of business and finances, and to a somewhat lesser extent in multi-agent systems. There have been only a few attempts to apply it to robotics. Miura and Shirai contributed a method of “parallel scheduling of planning and action for a mobile robot” [41] that exploits the advantages of planning during execution of actions, and acting during the planning phase. Their method proposes to probabilistically predict the outcome of an action while it is running, and proactively planning further actions based on the predicted results of the current action.

Armano et al. propose an architecture for autonomous agents for computer games [2]. The architecture consists of an arbitrary number of N layers with varying levels of granularity. Each layer has a deliberative, a proactive, and a reactive module that communicate with each other and with their corresponding modules on the next higher and lower level. The proactive modules are responsible for creating and refining plans based on the commands received by the deliberative modules, and for passing the plans on to the reactive modules to execute them. The

Figure 2.17: Proactive communication is exchanged between these Sony AIBO robots under the BITE architecture [30].

described proactive behavior is not proactive in our sense as it represents conventional symbolic planning, with re-planning taking place in case of a break command by the lower reactive modules. A more detailed description of the proactivity of these modules is not given, and the whole proposal seems to be at most the beginning stage of a project.

A model for an agent architecture is presented in [20]. Its main contribution is called adaptive goal prioritization and consists of proactively adjusting goal priorities when conditions change in dynamic environments to achieve an overall higher satisfaction level; this requires a model of the dynamic environment. The method, however, is more related to learning than to our understanding of proactivity as acting without explicit commands. The unified planning and execution framework IDEA [43] equips agents with a planning module that is invoked reactively during the normal execution cycle. In case the agents foresees any problems in the future, it can choose to invoke the planner proactively to alleviate those problems. There is no description of how this problem prediction is done, however, and how the problem circumvention may look like.

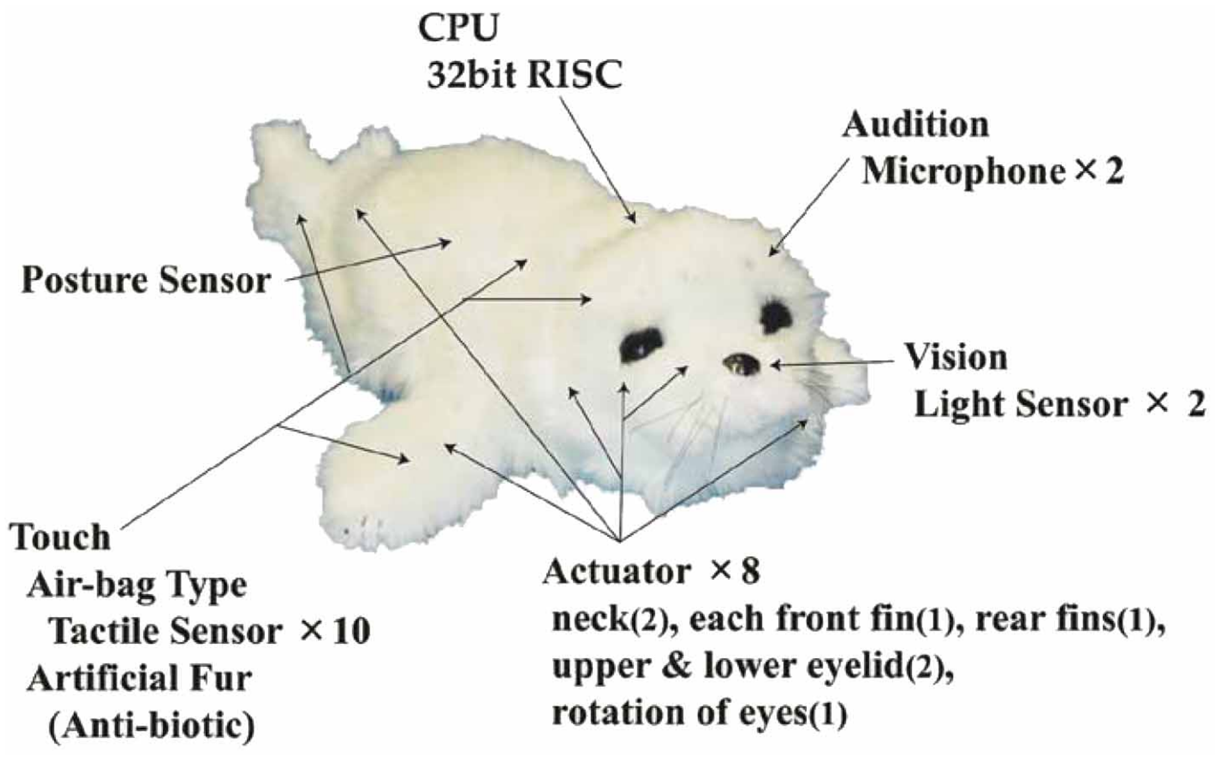
A framework that aims at automatically controlling the teamwork in teams of behavior-based robots was introduced in [30]. It works by maintaining the three hierarchies of organization, task/sub-task behavior, and social behaviors. As part of the organization hierarchy management, robots can proactively pass on sensed information to robots on sub-teams if it is relevant to them. This kind of proactive communication has already been demonstrated in the CAST architecture [71]. In order to test the framework with real robots, an instantiation of the it named BITE has been developed. In contrast to the robot used in this research, the employed Sony AIBO robots under the BITE architecture (cf. Fig. 2.17) are behavior-based and thus not able to execute actions as complex as in our case.

Proactive path planning is introduced in [38]. The proactivity consists in the robot's taking into account possible mobile objects that could interfere with its path from regions that the robot's sensors cannot perceive. In such cases of blind regions, the robot computes a velocity profile that guarantees that the robot is able to stop in time before a collision can occur.

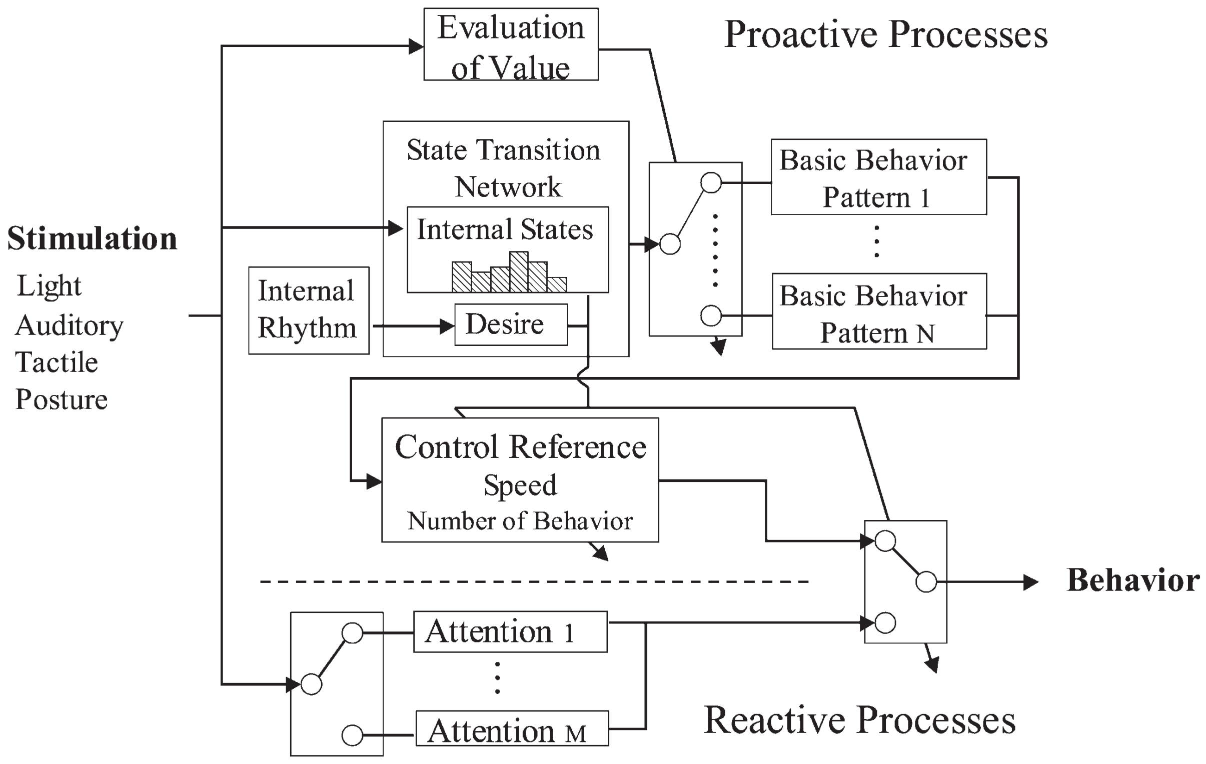
De Croon et al. examine how embodied and situated agents can develop proactive behavior [16]. Proactive behavior here is distinguished from purely reactive agents with sensory-motor skills only in that proactive agents accumulate sensory-motor information over time and in this way generate internal states. Their motor behavior then is a function of both the current environmental state and the internal state. The actual nature of this behavior is of no concern, however, but rather its ability to change and adapt over time. The internal state is realized using neural networks, comparing different varieties such as feedforward and recurrent neural networks. Thus it is impossible to extract any human intelligible information about the internal state, prohibiting any meaningful analysis or goal-directed adaptation.

One of the few attempts to directly apply proactive execution of tasks to robotics and human-robot cooperation is the plush seal robot named Paro [65, 66]. An image of Paro along with its sensors and actuators is depicted in Fig. 2.18(a). The authors differentiate between three types of behaviors of Paro: Proactive, reactive and physiological behaviors, see Fig. 2.18(b). The proactive behavior in turn is generated by two layers: the behavior-planning layer and the behavior-generation layer. As part of the behavior-planning layer, Paro features a set of internal states whose numerical levels are changed by external stimuli via its light, auditory, tactile, and posture sensors and decay over time. Paro also possesses a desire that is controlled by its internal rhythm. Both the internal states and the desire determine the current basic behavior pattern through a state transition network. These basic behavior patterns are limited to different poses and movements. The behavior-generation layer receives these basic behavior patterns and modulates them in terms of speed and number of repetitions based on the internal states and their variation. Moreover, this layer shifts the priority between proactive and reactive behaviors based on the internal states. Finally, there is a long-term memory that supports Paro's reinforcement learning which is designed to prefer human stroking behavior over beating. In contrast to this proactive behavior, the reactive behavior consists of immediate reactions to sudden stimuli such as looking into the direction of a perceived loud sound. In summary, the authors emphasize that the term "proactive" is somewhat misleading in that Paro's proactive behavior tries to simulate that of a real seal and therefore is very primitive compared to the proactiveness of a human being.

An "embodied proactive human interface" named PICO-2 (Proactive Interface for Communication) is described in [35]. The idea of this proactive interface is shown in Fig. 2.19(a). Basically, two humans communicate remotely over a phone line, but instead of a telephone, a humanoid robot of the type HOAP-2 (*Fujitsu Automation Ltd.*) is replaying the remote person's voice messages and image, depicted in Fig. 2.19(b). Furthermore, the robot executes gestures and motions that are supposed to convey the remote user's intention. Due to the employed

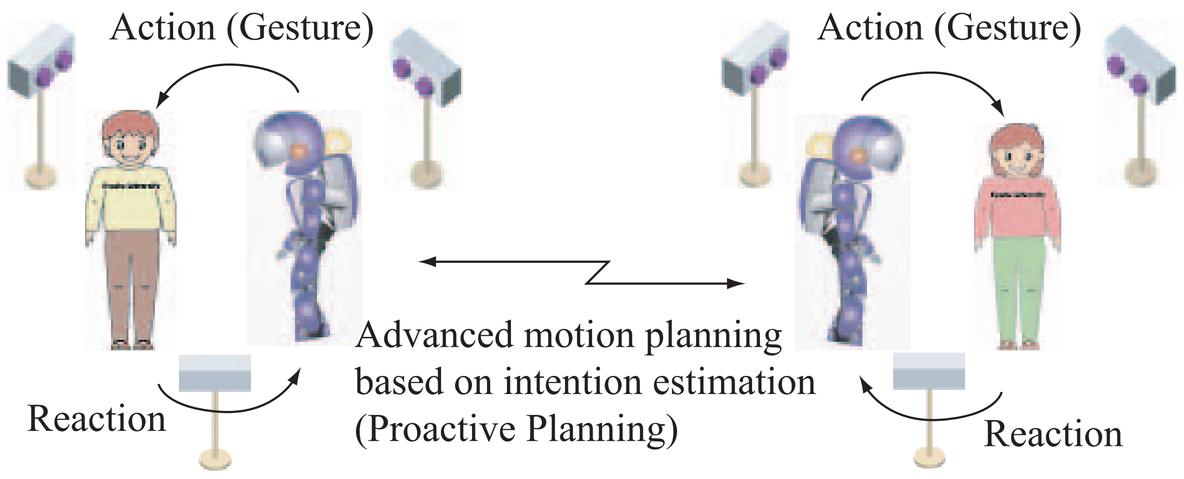


(a) The sensors and actuators of Paro.

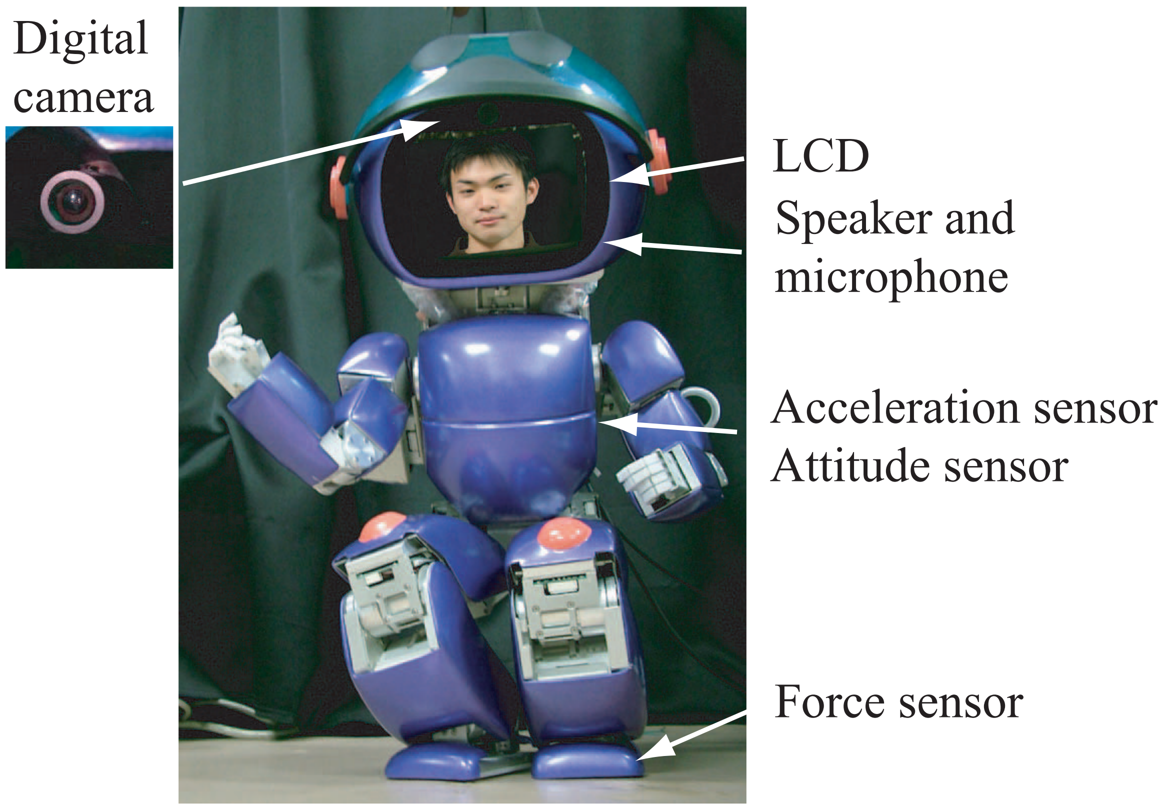


(b) Paro's behavior generation system.

Figure 2.18: The seal robot Paro [66].



(a) The concept of the proactive interface.



(b) Image of the interface.

Figure 2.19: The embodied proactive human interface "PICO-2" [35].

gesture-based intention recognition of the involved persons, an explicit command to control the respective remote display robots is not necessary. Rather, the recognized intention is displayed in the robot's own gesture skills. The robots thus work as AVATARs in order to improve the human-human communication. Due to the combination of the estimation of the human intention and the subsequently proactively executed robot action, the PICO-2 system is related in its overall concept to the work presented here. However, there are vital differences. For one thing, the robot actions are limited to gestures and similar motions, not arbitrarily complex skills. The bigger difference is the fact that this PICO-2 system does not represent human-robot cooperation in a strict sense, as each robot does not interact with the person in front of it, but rather only displays the remote user's gestures.

All in all, the paradigm of robots acting proactively when cooperating with a human being is still rather novel, and there are not too many published research projects in this area so far. When we use the term “proactive” in this context, we mean that the robot behaves in an autonomous way in the sense that the robot acts not only based on explicit commands by a human, but also based on implicit commands that arise from the human's estimated intention, the state of the environment or of the robot itself. Such a behavior has a strong impact on the perceptive and information processing requirements of a robot—it needs to sense the humans and the environment around him and extract the relevant information to allow its task planner to make an informed decision on what actions to execute. Furthermore, the issue of human safety and control over the robot grows along with the level of robot autonomy. These might be some of the reasons why this research area is only in its beginnings; hopefully this present research contributes to its progress.

3 Robot Architecture for Intuitive Human-Robot Cooperation

This chapter describes the control architecture for a service robot in a human-populated environment that was designed to accommodate the tactile language and the proactive execution interface. The main design goals were a natural and intuitive human-robot interaction and the ease of use for non-technical users. Moreover, we put emphasis on making the robot act dependably and predictably. We achieve these goals by employing a three-layer control architecture, with the focus being on the planning and sequencing layers. The essential design of these layers is presented in detail such that it becomes apparent how the architecture was actually implemented.

Developing a humanoid service robot for a human environment is an undertaking that presents challenges in many areas. An example for such a project is the collaborative research center "Humanoid robots—learning and cooperating multimodal robots" [5] that this research is part of. The goal of the research center is to create a humanoid service robot that is able to assist and collaborate with a human being in a kitchen environment [11]. Our part of this research effort is concerned with intuitive human-robot interaction.

The robot we used as our test bed is a 7 degree-of-freedom (DOF) humanoid manipulator shown in Fig. 3.1. It is equipped with tactile sensors [33], a stereo camera system, and we also integrated a dialog manager developed by another group within our research center [61].

Our design of the robot's control architecture (section 3.2) reflects the need to integrate these man-machine interfaces (MMI) and the requirements they imply. It also incorporates the intention recognition and proactive execution modules to provide a natural, non-verbal, and intuitive way for non-technical people to command the robot (section 3.3). Moreover, the design takes great care that the robot always acts dependably and trustworthily, a crucial requirement for getting accepted in a human environment. Since we operate in a real-world scenario, our execution supervisor (section 3.4) must take the possibility of exceptions and unplanned hangs into account and cope with them appropriately.

3.1 Motivation

When designing this robot architecture for a humanoid robot, we first put together a set of requirements the architecture must meet. We derived these requirements by analyzing several benchmark scenarios. Consider for instance a scenario where both the robot and a human are standing in the kitchen. The human then asks the robot to get him the cup of water from the

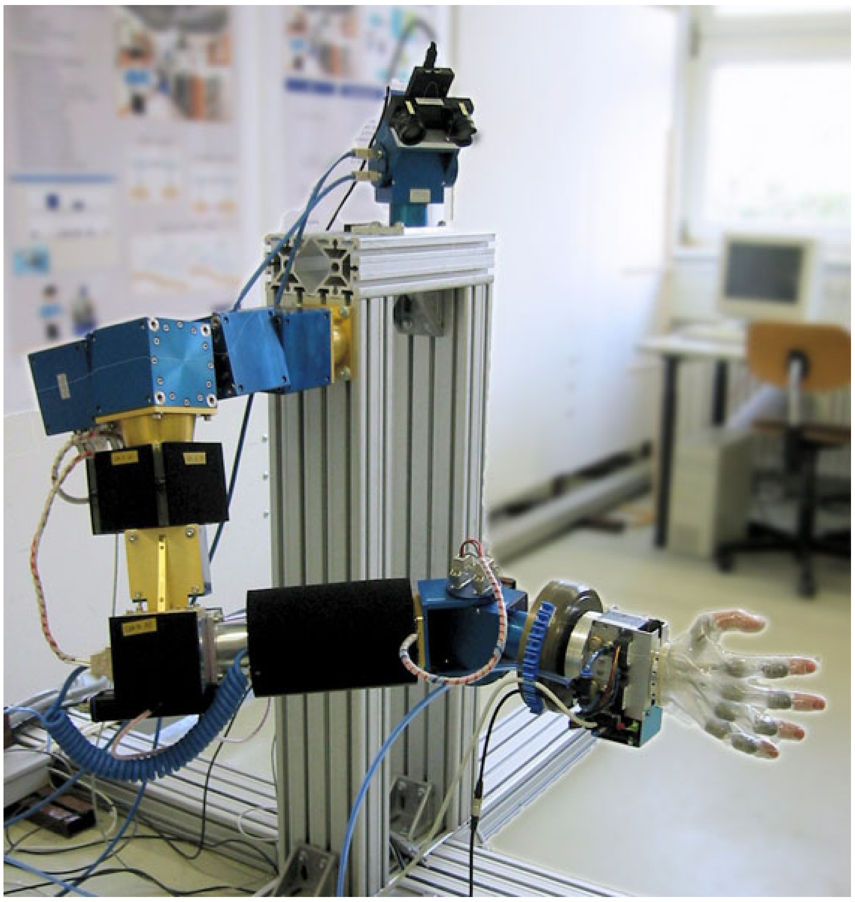


Figure 3.1: The humanoid manipulator.

kitchen counter. The robot acknowledges the command, moves to the counter, grabs the cup, brings it to the user, and hands it over to him.

Looking at such a scenario we identified the following requirements:

- The robot needs to be able to deal with an unstructured, dynamic, real-world environment. There will be people moving around and relocating objects like chairs or cups.
- The integration of a variety of man-machine interfaces must be feasible to allow for multimodal interaction with the robot. Sometimes it is more natural for a human to express a command verbally, and sometimes a gesture or a touch suffices.
- The interaction with the robot should be as intuitive as possible, especially for non-technical users. This demands in our view not only a natural language system, but also the possibility for implicit communication through intention recognition.
- Since this architecture is for a humanoid household robot, we must ensure human safety. Unlike an industrial robot, it shares a common workspace with humans. This means we must make a trade-off between operating speed, efficiency, workload capability, and safety.

- To be accepted in a human environment, the robot needs to act dependably, reliably, and predictably. The human needs to feel in control at all times and be able to tell what will happen next. This further enhances human safety as well.

Additionally, we set ourselves these goals:

- The architecture should be easy to apply or port to other robots, and it should be modular to allow for extensions.

- The implementation of this architecture should be easy and straightforward. A humanoid robot is an arbitrarily complex system, and we need to make sure it remains maintainable, serviceable and manageable.

3.2 Architecture Overview

The robot architecture describes the basic building blocks that make up the robot system and the relations they have with each other. In addition, it defines the interface of the robot system to the human being that interacts with this robot. It is depicted in Fig. 3.2.

The robot system's interface to the outside world is composed of its *Sensors* and *Actuators*. The *Actuators* may include arms, hands, a head, a torso, and a mobile platform or legs. In the case of our robot, it is the aforementioned 7-DOF arm, an 10-DOF pneumatic hand, and a pan-tilt unit as a neck or head.

On the *sensor* side we employ a stereo camera as visual sensor, a close-range microphone for natural language commands, a force-torque sensor at the wrist, and tactile sensors that cover a substantial part of the robot's surface as an artificial skin.

As to the internal control structure, we opted to go along with the popular concept of a three-layer architecture [17]. The three layers it consists of are the *Task Planner*, the *Execution Supervisor*, and the *Task Coordination & Task Execution* layer.

The *Task Planner* interfaces with the outside world through the *Man-Machine Interfaces* and the *Intention Recognition & Proactive Execution* modules. These make up the input on which the *Task Planner* selects the *task* to be executed or aborted (see section 3.3 for the details). The selected task (e.g., a pick-and-place operation) is passed on to the *Execution Supervisor* module.

The *Execution Supervisor* module's responsibility is to enable the selected task, abort it if necessary, and to report back any relevant signals to the *Task Planner* that might occur during execution, such as error or completion messages. These mechanisms are described in section 3.4. The tasks are coded as finite state machines or Petri nets and invoke low-level subtasks, such as move_to_x or grasp_object, as necessary. We call these subtasks *actions*.

The *Task Coordination & Task Execution* layer is comprised of the actions that represent closed-loop feedback control circuits controlling the actuators based on the sensory information

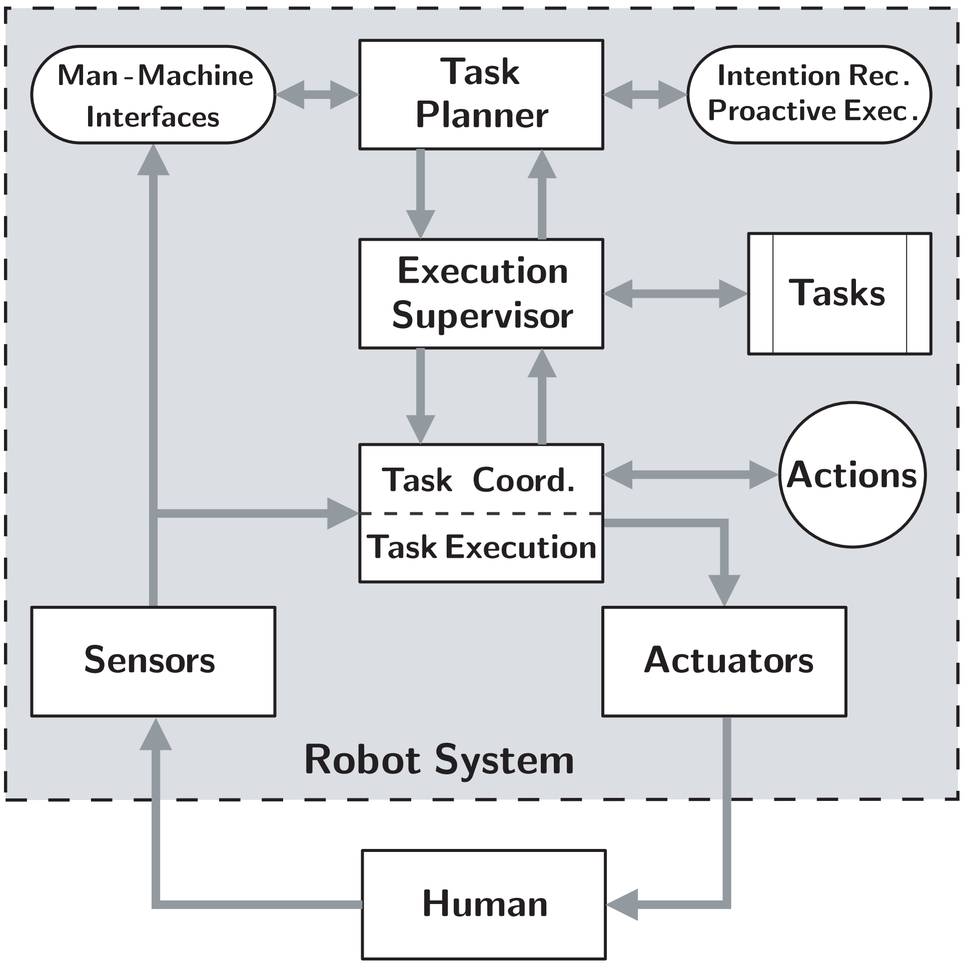


Figure 3.2: Robot control architecture.

from the external world and the internal status through angle transmitters and strain gauges. As this layer is highly dependent on the respective hardware, we won't elaborate on this architecture layer here any further.

As a software framework for the architecture we use the Modular Controller Architecture (MCA2) that runs both under Linux and Windows [52, 53]. Its predominant paradigm is that of a controller with sensory and control I/O. Each control *module* receives sensory input from lower-level modules and returns control output to them. It passes on sensory output to higher-level modules (possibly after some processing) and receives in turn control input. Modules are connected via these I/O-edges, and multiple modules can be grouped together. One or more modules and groups are then collected in an executable *part*. Parts communicate through TCP/IP, across platforms if necessary. Real-time requirements are met by specifying the cycle time parameter of the respective modules. Thus the less time-critical higher-level layers of the robot architecture can be run at lower frequencies than the closed-loop control modules of the lowest level. An administration tool that visualizes the module hierarchy and a GUI tool for easy user control complete the framework.

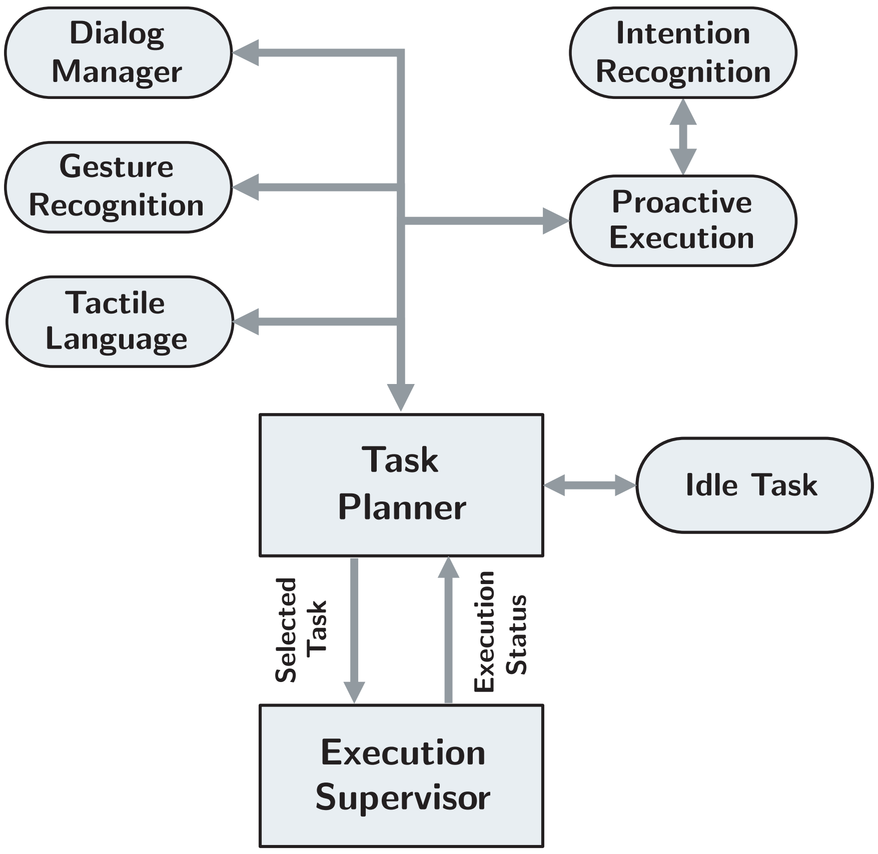


Figure 3.3: Task planning layer.

3.3 Task Planner

The task planning layer is responsible for selecting which task to execute at any given time. This selection is based on the input of the man-machine interfaces (MMIs), the proactive execution, and an idle task, see Fig. 3.3.

Since the service robot must act dependably in a human environment, it is crucial that the task selection works in such a manner. Namely, we need to make sure that we select the current task in a deterministic way, and that the selected task corresponds to the intention of the human user. In contrast, a task selection mechanism based on some intrinsic, predefined drives [64] of the robot is unacceptable. As an example, imagine that a human being addresses the robot and asks it to get a beverage from the fridge. Then the expected behavior of the robot is to fetch the requested item right away, not to pick up a piece of lint from the carpet. If it is currently executing some other task, we expect it to verify that it should abort that other task and start executing the new task. To achieve this, our task selection process follows a clear hierarchy:

- The top priority is given to the MMIs such as dialog manager, gesture recognition, or tactile language. Among these MMIs, the dialog manager again gets the highest priority. If a

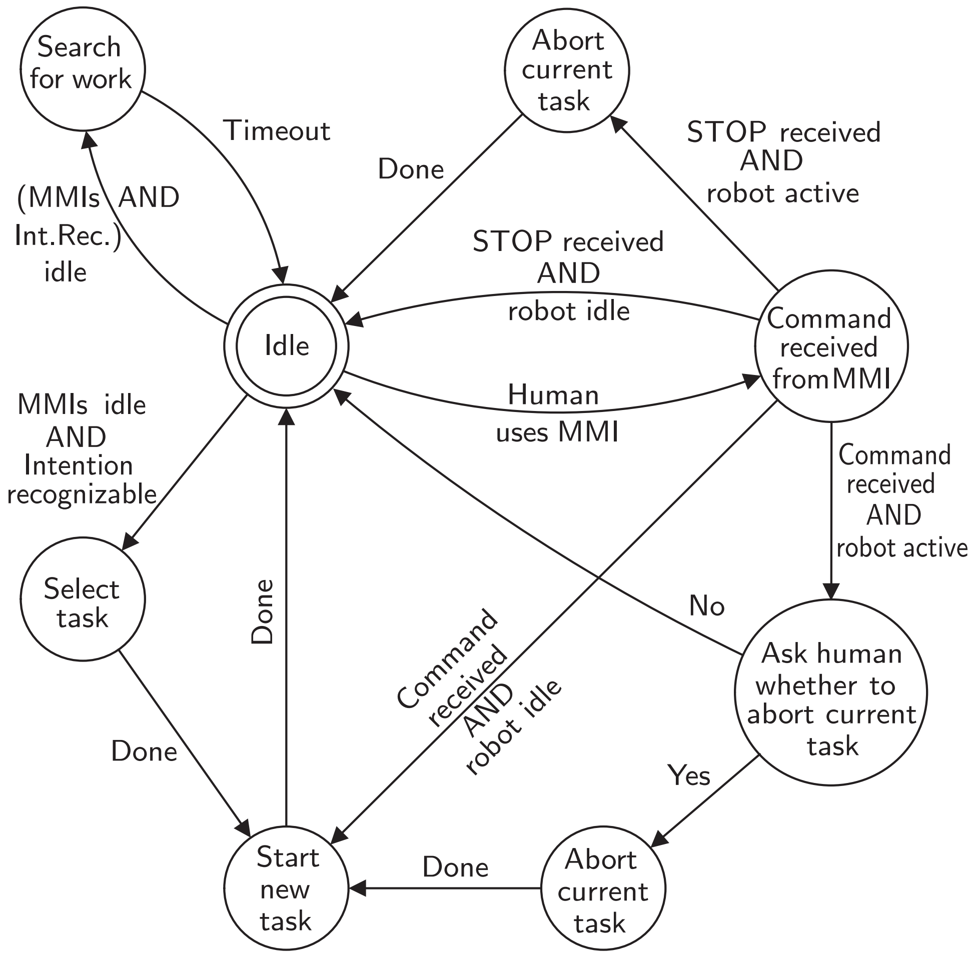

Search for work
Abort current task
Timeout
Done
STOP received AND robot active
(MMIs AND Int.Rec.) idle
STOP received AND robot idle
Idle
Command received fromMMI
Human uses MMI
MMIs idle AND Intention recognizable
Command received AND robot active
Select task
Done
No
Command received AND robot idle
Ask human whether to abort current task
Done
Yes
Start new task
Done
Abort current task

Figure 3.4: Task selection process of the task planner.

command is received through an MMI, it takes precedence over any other task selection.

- If no command has been received through an MMI and the robot is idle, we check if the proactive execution module can assist us in determining a task that is likely to correspond to the human's intention.
- If the proactive execution fails to provide a useful output, an idle task is started. For example, this task could make the robot wander around the premises and search for work to do, using the above two methods. If the robot is unable to contract any task, it would go into a standby mode at a predetermined safe place.

The finite state machine realizing this behavior is shown in Fig. 3.4. Note that in case we receive a new command through one of the MMIs and the robot is still busy executing another task, we always check back with the user via the dialog manager to determine whether to stop the currently active task or ignore the new command. We don't queue up commands to keep things simple. If we had a command queue, we would be required to re-sort this queue upon the issue of a new command according the relevance of the latter at runtime, and to come up with a scheme to determine the relevance or priority of each command to make the sorting possible at all.

3.3.1 Man-Machine Interfaces

MMIs provide *explicit commands* from the user. There is a arbitrary number of MMIs that can be conceived. In order to handle them, we use our goal of intuitive human-robot interaction as a guideline to prioritize the actually employed MMIs. Specifically, we feel that a dialog manager that is capable of handling natural speech is not only intuitive but also quite unambiguous, thus providing us with clear commands [61]. Therefore we give the highest priority to the dialog manager. The tactile language that we conceived and that is presented in chapter 4 is of equal importance to us. It provides a command interface that is both redundant and complementary to the dialog manager and has been proven to be very reliable and practical; its priority was therefore set right below the dialog manager.

For the users' convenience we can accommodate any other MMI in this mechanism. The MMIs are being evaluated as the user engages them. Should more than one MMI be used at any time we ignore all MMIs but the one with the highest priority, providing for an easy conflict resolution.

3.3.2 Intention Recognition and Proactive Execution

This section focuses on the collaboration between task planner and proactive execution; chapter 5 treats intention recognition and proactive execution in full detail. To understand the concept of intention recognition and proactive execution, consider the host of a party walking around with a tray in one hand and a decanter filled with his favorite beverage in the other. When approaching a guest holding an empty glass the party host decides that the guest would probably welcome a refill (*intention recognition*) and thus presents the decanter to the guest (*proactive execution*). Based on the guest's reaction the host either goes ahead and fills the glass, or presents the tray so the guest can get rid of his empty glass.

In our implementation, the intention recognition provides the task planner with a list of currently valid intentions and its probability density. These intentions must be well-known to both task planner and intention recognition. When the robot is supposed to act in response to the input from the intention recognition we have to distinguish several cases:

- The first case is that no intention can be inferred from the available sensor data, which is equal to all intentions having the same probability.
- When there are two or three candidates as likely estimates for the human intention we have the chance to make a guess about the "true" intention.

- The last case happens when there is indeed one single intention that obviously dominates the rest. This is the ideal case, as it gives the task planner a clear idea of what task to execute.

The last case is the easiest case to handle. The proactive execution module chooses the appropriate task that is associated with the recognized intention and forwards it to the task planner. Thus, given there is no MMI input at the same time, the robot acts according to the recognized intention. In the case where no intention was recognized with a sufficient certainty, the proactive execution module will not suggest any task, and therefore the task planner would select the idle task as described earlier.

The remaining second case is harder to deal with. For the event that there are two or three plausible intentions to choose from, we developed the concept of the *proactive execution* of a task. This means that instead of idling we pick an intention and pretend that this is the wanted intention, and suggest an appropriate task to execute. To indicate this event to the task planner, the proactive execution module turns the signal `PE_Flag` on.

In any case, the proactively suggested tasks are by design based on the estimated human intention and thus represent *implicit commands*. As such, they are afflicted with uncertainty and need to be treated specially; this is done in the execution supervisor (section 3.4).

3.3.3 Idle Task

The idle task's main purpose is to provide for a default control mode that becomes active when there is nothing else to do. This is the case whenever a task has been finished, no new command is being provided by the user, and no intention can be inferred from a user in the vicinity. The default control mode allows us to have the robot in a well-defined state executing a well-known task at any given time. We believe this is obligatory to render a service robot dependable.

Additionally, the idle task serves to enhance the usefulness of our robot. Instead of passively waiting for an input it actively searches for work to do. When meeting a person on the way it tries to get her attention and offer its services. Should the robot fail to encounter anybody that needs help within a timeout period it will go into an energy-preserving standby mode at a safe place that has been programmed previously. This way we make sure that we don't squander our resources in the absence of meaningful tasks to do.

3.3.4 Exception Handling

An exception occurs when a task finds itself unable to continue execution. The reason for that can be an unknown situation or a known situation that the task isn't able to handle. This happens, for example, when the robot is carrying a cup of water and accidentally drops it on the floor.

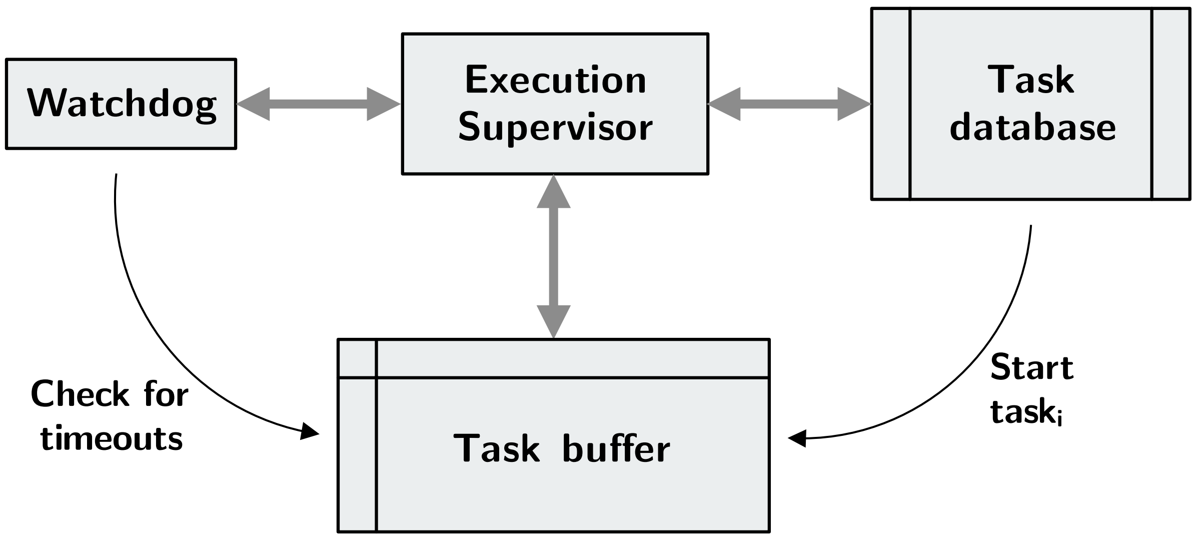


Figure 3.5: Execution supervisor layer.

Due to the paradigm of abstraction that is underlying our layered architecture (see Fig. 3.2), all the knowledge about how to execute the tasks in different possible circumstances is located in the execution and task coordination & task execution layers. Therefore, if these layers fail to complete a task, the planning layer has only two general, high-level alternatives for dealing with such an exception. The default one is to use the dialog manager and let a human user decide what to do next.

For the event where there is no human around to consult, or the resolution of a situation involves actions that are not available to our robot, the concept of telepresence [46] comes into play. The idea behind telepresence is to have a human operator who is located in a service center and who can operate our robot remotely by appropriate interfaces such as data gloves, a head-mounted display, and a haptic feedback device. Hence, for a second alternative, we trigger the telepresent operation of our robot and have the human service operator clear up the situation.

3.4 Execution Supervisor

The execution supervisor layer (cf. Fig. 3.5) receives the selected task and any related parameters as an input from the task planner. It then decides whether to start the task or discard it, and informs the task planner of the execution status. In order to explain this process, the nature and properties of tasks will be introduced first.

3.4.1 Task Specification

Tasks are specified as deterministic finite state automata (DFAs) or Petri nets; in a more low-level view of the software architecture, they are also called *skill objects* [18]. Depending on

Name	ArmExecObject	Resources	Timeout	Description
inithead	IdleExecObject	`0100100`	-1	Initialize head
startrarm	ArmExecMoveStartPos	`0000001`	30	Start right arm
guide_arm_ua	ArmExecCont_ua	`0001001`	-1	Guide upper arm
follow_human	IdleExeObject	`0100100`	-1	Follow human
gesture_prismatic5	Grasp_Prism_5_Fg	`0000010`	-1	Grasp Prism. 5 Fg.

Table 3.1: The task information database (excerpt).

their control domain, they can be *arm execution objects* or *hand execution objects* in our implementation. Starting a task means to signal either the arm or hand planner which execution object to start, and the respective planner then instantiates this object in a shared memory area which is called *black board* within the MCA2 framework. For this mechanism to work, both the execution supervisor on the one hand and the arm and hand planner on the other need a common task information database. An excerpt of this database is shown in Table 3.1. It contains the following entries:

- A unique task name that is used to index this table.
- The arm or hand execution object that this task is mapped to. This mapping tells the arm and hand planner what skill object to instantiate.
- A resource vector indicating all resources needed to execute the task. These resources include all sensors and actors that are potentially shared between tasks. At our current project status, there are seven resources that the vectors are comprised of:
  - ∗ force-torque sensor
  - ∗ stereo camera
  - ∗ tactile sensors (complete image frames)
  - ∗ tactile sensors (resulting forces)
  - ∗ head control
  - ∗ hand control
  - ∗ arm control
- A timeout value for the watchdog timer indicating a maximum running time of the task. This is recommendable since, for example, in case of an undetected exception the execution of a task could be stuck forever. Such a halt, however, is obviously undesirable: (a) the task itself doesn't make any progress, (b) potential new tasks that could do some useful work are blocked, and (c) it might also pose a safety risk. For some tasks, though,

Index	Task ID	Command	Req #	Parameter 1	2	3	Resources	Timeout	ExecTime
0	1010	inithead	4	0.0	0.0	0.0	`--x--x-`	−1.00	−1.00
1	1008	startrarm	7	0.2	0.0	1.0	`x------`	30.00	26.70
2	0	empty	0	0.0	0.0	0.0	`-------`	0.00	0.00
3	0	empty	0	0.0	0.0	0.0	`-------`	0.00	0.00

Table 3.2: The task buffer.

such a timeout value doesn't make sense. Think of the task `'guide arm'` where the human positions the robot arm through the tactile sensors in zero-force-control mode—the robot cannot know in advance how long the human wants to move the arm. For these cases where the task is supposed to run indefinitely, a value of -1 can be entered in the database. To actually alleviate a potential hang we employ a watchdog in the execution supervisor that checks in each execution cycle whether a task's execution time has exceeded its timeout limit or not. In the event of a timeout limit excess, the respective task will be aborted. To that end, a task should contain a safe state it can transition to. In case the robot cannot switch immediately to its safe state (imagine the robot holding a pot of coffee), some intermediate states implementing some kind of rollback will be required as well (i.e., the robot replaces the pot on the kitchen counter).

- A human-readable task description.

3.4.2 Making Proactive Execution Work

There are several problems related to proactive execution that have to be addressed by the execution supervisor:

- The probability density over the list of intentions is subject to change. If this change happens too quickly we might end up starting and aborting tasks in rapid succession without getting anything done.

- Any task that corresponds to one of the intentions we monitor must be interruptible. This is due to the very nature of proactive and thus tentative execution, and the just mentioned inherent changes of our uncertain intention information. Moreover, a rollback of actions might be necessary.

- We execute tasks proactively to provoke some clarifying reaction of the user, which we can in turn use to reduce our uncertainty about the user's "true" intention. Thus the corresponding tasks should be geared towards triggering a useful human behavior response.

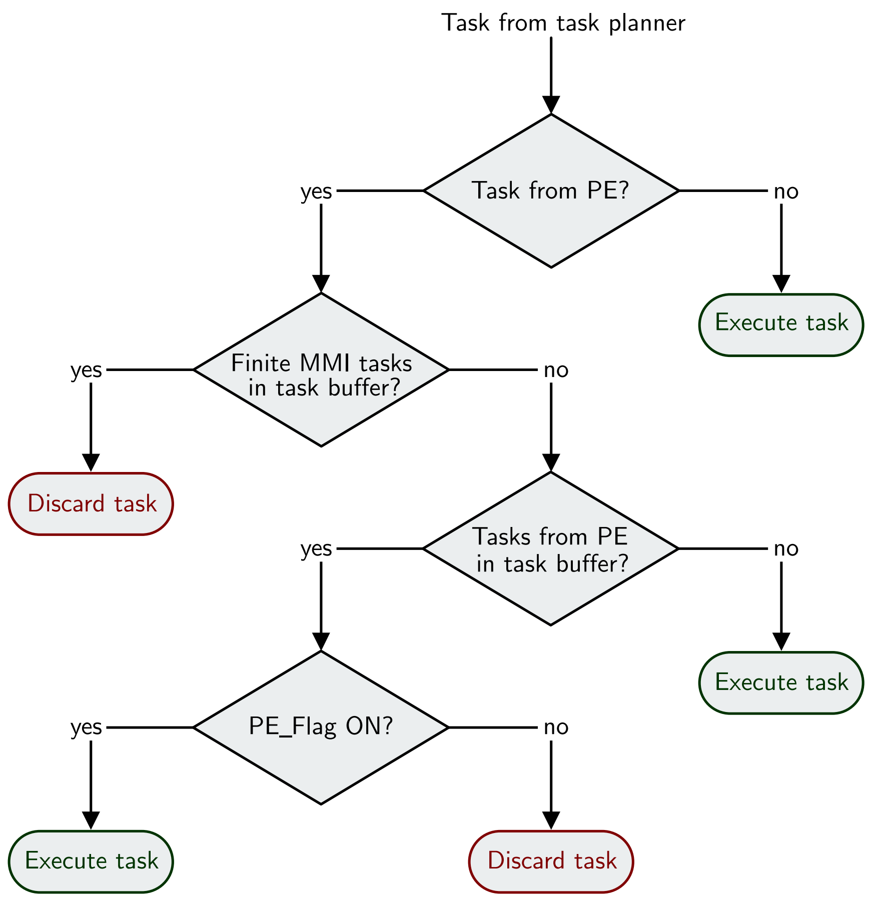


Figure 3.6: Decision algorithm for starting or discarding tasks.

To handle the first issue, a decision algorithm whether to start a task or discard it was devised (see Fig. 3.6). It makes use of the list of currently executing tasks, in the following called *task buffer*. It is shown in Table 3.2. The decision algorithm works as follows: Whenever the execution supervisor receives an new task from the task planner, it tests whether it was issued by an MMI or by the proactive execution (PE) module. In the former case, the new task is simply started. If it needs a resource that is used by a currently running task (indicated by the `Resources` column in the task buffer), this older task is aborted since it is assumed that the human knows what he is doing. In the latter case, it depends whether there is an MMI task currently running that has a finite duration ($Timeout \neq -1$). If that is true, that task takes precedence, and the PE task is discarded. Otherwise, the new PE task is executed if there is no other PE task in the task buffer. If there is, though, it will still be executed and possibly abort the old PE task if the `PE_Flag` is on (see section 5.9 for more details). The reason is that in case of tasks that have been proactively executed, we can assume that the new task reflects a better human intention estimate and replace the current task. Should the `PE_Flag` be off, however, the new task is discarded to allow the current task to finish.

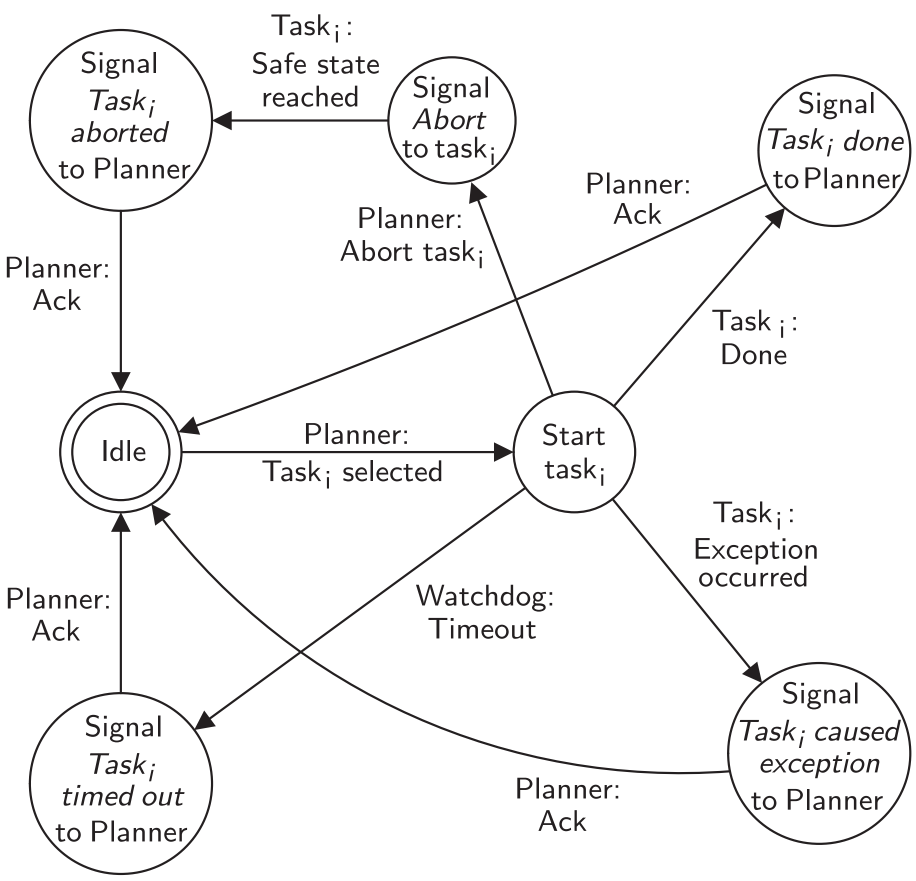


Figure 3.7: Execution supervisor finite state machine.

The second issue is already taken care of by the same mechanism that we use to abort a task due to a new user command, after a timeout and in case of an exception. We transition to a safe state, possibly via a series of intermediate states if we need to perform a rollback.

As to the last point, we need to make sure that we associate different intentions with tasks that are virtually orthogonal to each other. Recall the illustrative example from section 3.3.2. In our opinion it is most intuitive for a human to be physically presented with the object of interest. The human can then assume an inviting posture if he agrees, or move backwards to decline. Thus, only nonverbal communication is necessary to resolve the uncertainty. In addition to this initial approach we plan on further investigating the optimal choice of tasks in the near future.

3.4.3 Task Execution Supervision

Once a task is started and running, it is entered in the task buffer. The actual task execution supervision procedure is outlined by the state machine in Fig. 3.7. It details how the execution supervision works and the different execution outcomes that can be distinguished:

- If the task executes and finishes normally, it returns a `done` signal. The execution supervisor forwards this signal to the task planner and returns to idle.
- If an exception occurs, the task will notify the execution supervisor and transition to a safe state. It is then up to the task planner to decide on how to proceed. In the current implementation, no further action is taken.
- In case a task times out, i.e., it fails to make any progress within a timeout period, the watchdog will notice this as described earlier. The execution supervisor will send a `timeout` signal to the task such that it can assume a safe state. Again, the planning layer will decide on the next task to execute.
- Should the task planner send an `abort` command, the execution supervisor will forward this command to the running task and remove it from the task buffer. `Abort` causes the task to change to its safe state. Once this safe state is reached, the task will raise a flag to let the execution supervisor know, which in turn signals the successful task abortion to the task planner.

3.5 Summary

This chapter presented a robot architecture for a service robot in a human-populated environment. We gave an overview of the general three-layer robot architecture, then highlighted the task selection mechanism in the task planner and its capability of accommodating arbitrary MMIs as well as the output of the proactive execution module. In addition, the functionality of the execution supervisor was detailed, including the task structure, the task buffer, the decision algorithm for starting or discarding tasks, and the task execution supervision mechanism.

4 Tactile Language

This chapter presents a tactile language for controlling a robot through its artificial skin (Fig. 4.1). This language greatly improves the multimodal human-robot communication by adding both redundant and inherently new ways of robot control through the tactile modality. We defined an interface for arbitrary tactile sensors, implemented a symbol recognition for multi-finger contacts, and integrated that together with a freely available character recognition software into an easy-to-extend system for tactile language processing that can also incorporate and process data from non-tactile interfaces. The recognized tactile symbols allow for both a direct control of the robot's tool center point as well as abstract commands like "stop" or "grasp object *x* with grasp type *y*". In addition to this versatility, the symbols are also extremely expressive since multiple parameters like direction, distance, and speed can be decoded from a single human finger stroke. Furthermore, our efficient symbol recognition implementation achieves real-time performance while being platform-independent. We have successfully used both a multi-touch finger pad and our artificial robot skin as tactile interfaces. The evaluation of our tactile language system was done using test persons that had to learn to language symbols, perform certain missions, and finally report on their experience using the NASA Task Load Index. A short description of the tactile language was first published in [48], while the detailed evaluation was presented in [49]. Implementation details are described in [26].

This chapter is organized as follows: The following section 4.1 motivates the tactile language. In section 4.2 we give an overview of the system and the hardware we use. The different input modes and the symbol recognition are described in section 4.3. Section 4.4 presents the finite automaton we use to decode the entered symbols and generate robot control commands. Finally, we present the results of the evaluation of our tactile language in section 4.5.

4.1 Motivation

When it comes to multimodal interfaces, most solutions include visual, auditory, or joystick-based haptic interfaces. While the latter are well suited for the direct control of the tool center point (TCP) of a robot but otherwise are rather limited in their usability, the former suffer from slow processing speeds and unnecessary overhead for simple commands. Our idea was to combine the advantages of the aforementioned interfaces into a new one that is capable of both matching and complementing them. To this end we designed a *tactile language* that affords to communicate with and control a humanoid robot through its artificial skin. Although this is

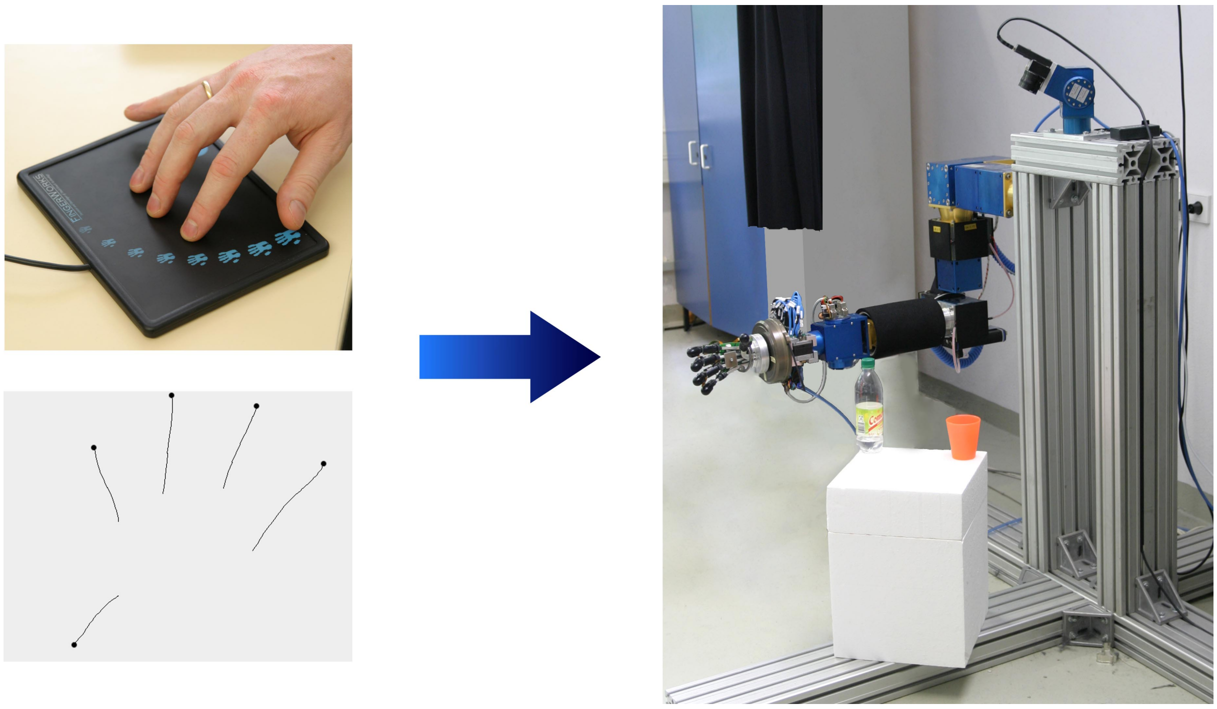


Figure 4.1: Tactile Language for Robot Control.

our application scenario, the tactile language is neither limited to a robot nor to artificial skin. Any tactile sensor with a spatial resolution will do as input device (see subsection 4.2.2 for the sensors we used), and instead of a robot other suitably intelligent machines are conceivable as communication partners.

Important design criteria for this tactile language were an intuitive usage, versatility, easy extendability, and low computational demands for approximately real-time capability. This lead us to incorporate several input modes, direct and symbol based, and include a general-purpose handwritten character recognition software besides our own custom-designed symbols.

4.2 Hardware and Software

4.2.1 System Overview

An overview of our tactile language processing system is shown in Fig. 4.2. One or more artificial skin (i.e., tactile) sensors generate streams of data via their associated C++ contact processing routines. A Java program processes these data streams by consolidating the measured contacts, tracking them, and determining the end of one symbol input. Upon reception of a complete symbol it executes its symbol recognition routines for the detection of lines and curves. Additionally, it calculates relevant physical features of the symbol such as its length, radius, area, and eccentricity. Moreover, the complete symbol trajectory is handed over to the optical character recognition program JMerlin that is specialized in recognizing handwritten characters. Finally, these findings are interpreted by a deterministic finite automaton (DFA). The DFA recognizes valid command sequences and generates appropriate robot control sequences. The latter are then communicated via a TCP/IP socket to the robot control programmed within the framework MCA2.

4.2.2 Tactile Sensors

iGesture Pad

One of the tactile sensors we use is the *Fingerworks*[1] *iGesture Pad* as depicted in Fig. 4.3. It consists of a capacitive sensor array with a contact area of 16.5 cm×12.5 cm. It is capable of detecting a proximity image of materials with a high conductivity or permittivity such as a human hand. An example of a proximity image is displayed in Fig. 4.4. Its firmware interpolates the sensor measurements of a contact with a resolution of 0.2 mm in x direction and 0.5 mm in y direction. This yields a resolution of 1560×1135 pixel. The firmware segments the contacts and calculates the center of gravity, size and form factor of each contact [69].

DSAMOD-6

The *Weiss Robotics DSAMOD-6*, shown in Fig. 4.5, is a resistive pressure sensor module of the same type as the black artificial skin on our humanoid manipulator that is visible in Fig.s 3.1 and 4.1. This particular sensor module consists of a 12×16 sensor cell array with a spatial resolution of 6 mm in both x and y direction. The output value of the sensor cells is not proportional to the pressure but rather follows a characteristic curve. The minimum pressure that can be measured is less than 2 kPa, the maximum is ca. 250 kPa. The tactile sensor *DSAMOD-6* is connected via the controller *DSACON32* to the computer. The controller samples the sensor matrix, digitizes the output values with a resolution of 12 bit and transmits them via the serial interface at 115.2 kbit/s. For more details see [68].

4.2.3 Tactile Data Preprocessing

When using a tactile sensor and controller such as the *DSAMOD-6* with the *DSACON32* whose firmware merely provides the measured value of each tactile sensor matrix element (or *pixel*) without any additional data processing, it is necessary to preprocess that data to determine essential tactile image features such as the *number of contacts* and their properties such as the boundaries of the *contact regions*, the location of the *centroid*, and the *eccentricity*.

We determine the number of contacts by looking for local maxima, i.e., pixels that have values greater than all others within a 3×3 pixel neighborhood. In order to identify the region of a contact, we use the watershed algorithm [55] that we modified to fit our needs: Starting out

[1]The company *Fingerworks* has meanwhile ceased operations.

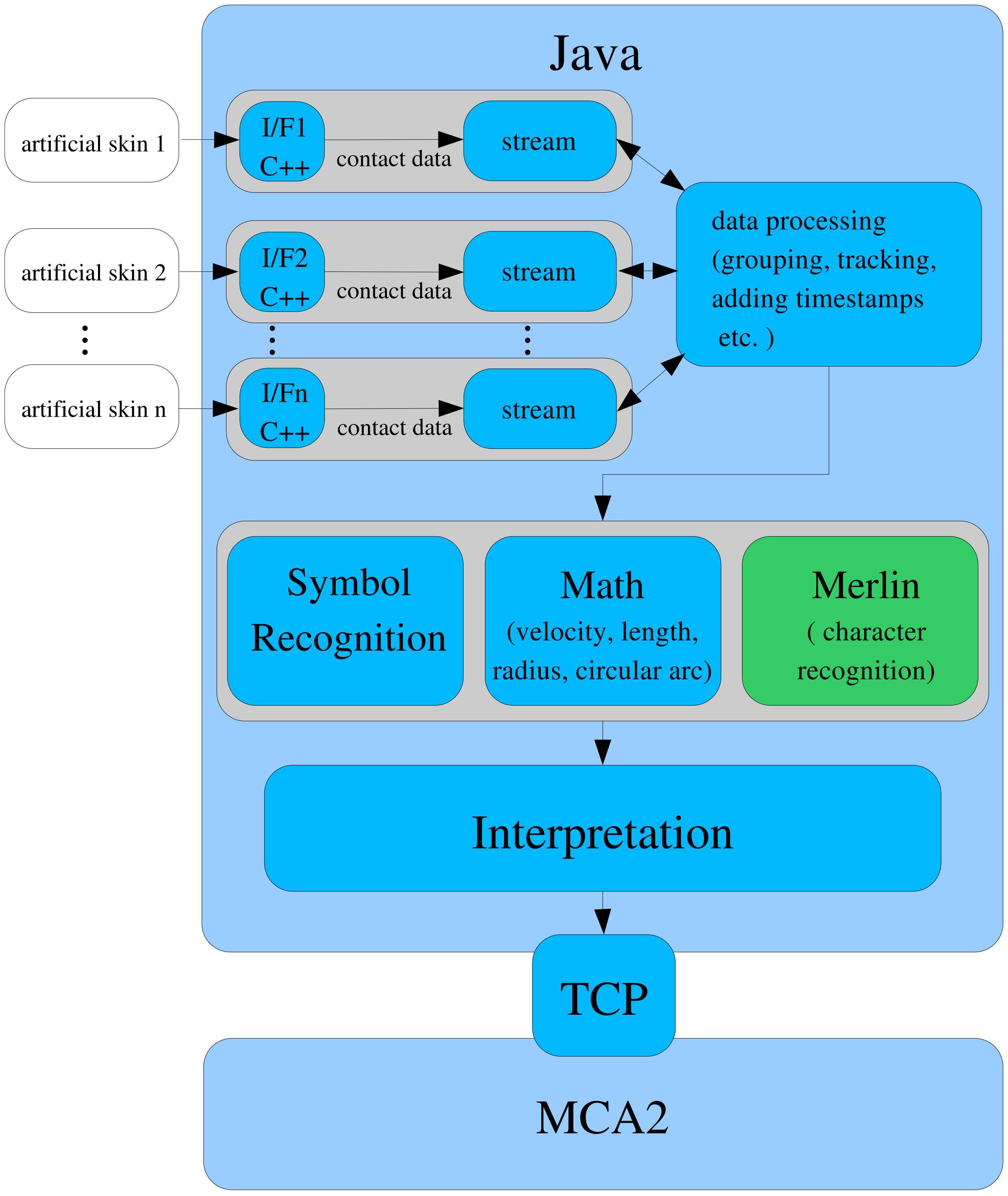


Figure 4.2: System overview with artificial skin interface, data processing and connection to the MCA2 robot control framework.

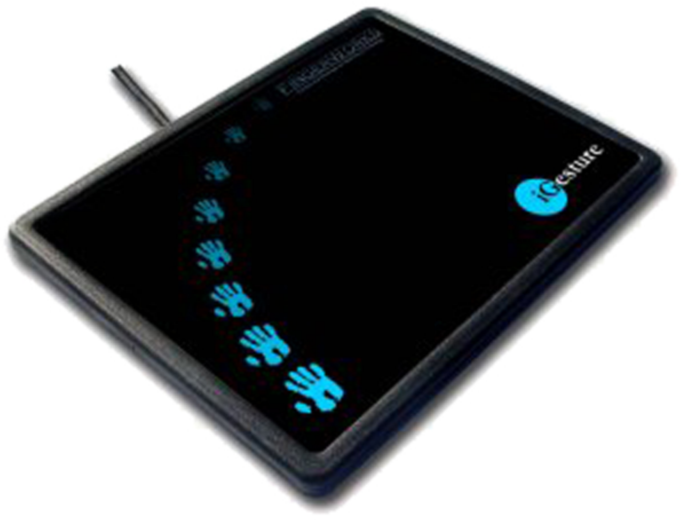


Figure 4.3: The *Fingerworks iGesture Pad*.

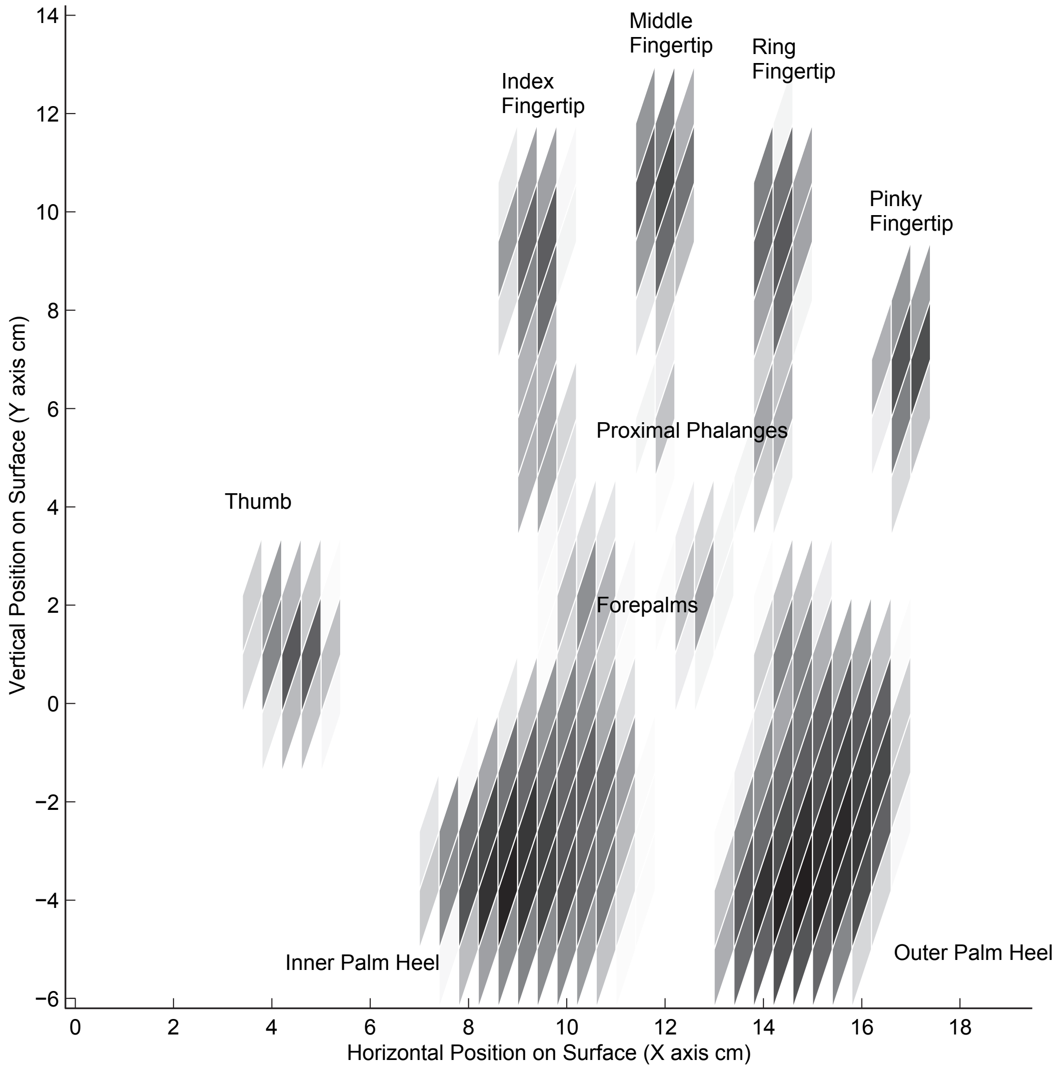


Figure 4.4: A proximity image recorded with the *iGesture Pad*'s multi-touch sensor technology [69].

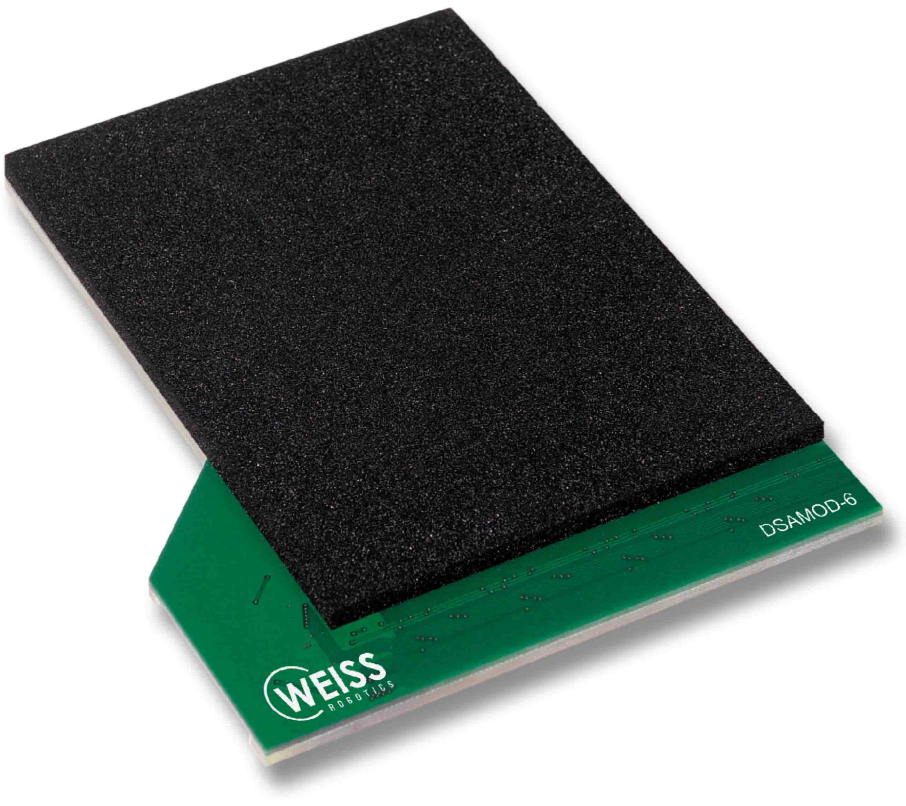


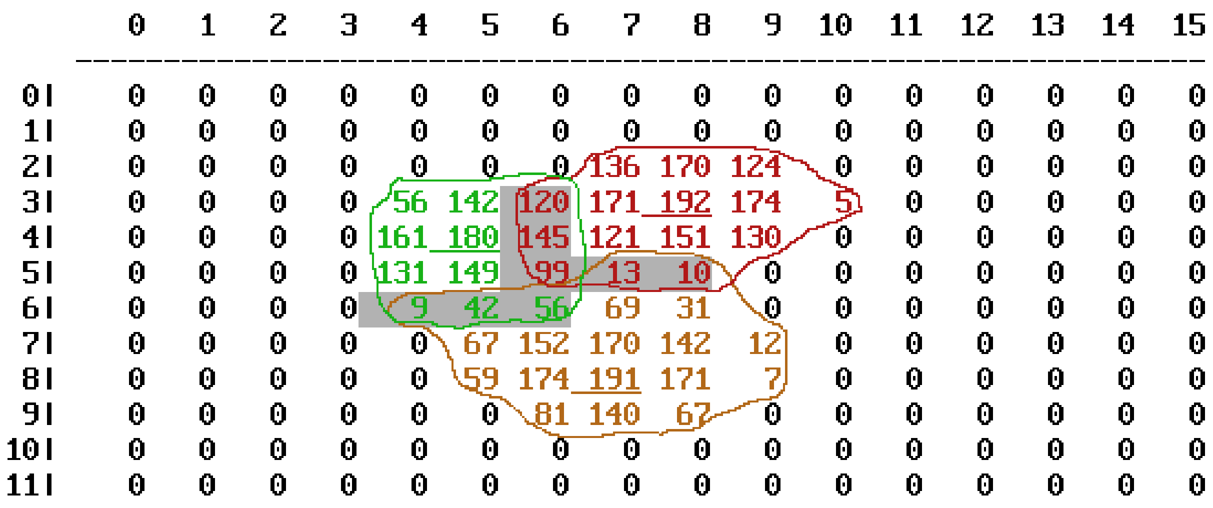
Figure 4.5: The *DSAMOD-6* resistive pressure sensor [68].

at the maxima, the “mountain tops”, water flows downwards. A new pixel is covered by water if there is a pixel in its 3×3 neighborhood that has both a greater or equal value and is already covered by water. If water flows from two or more sources merge at a pixel, this pixel becomes a “dam”, that is a boundary pixel between two or more regions, and no more water spreads from it. For the purpose of reducing aliasing effects with overlapping contact regions (see Fig. 4.6 for an example, with boundary pixels having a grey background) when calculating the center of gravity, these boundary pixels are made part of all adjacent regions with a modified value depending on their Euclidean distance from the maximum. If a boundary pixel $P(x_P, y_P)$ has an actual value of $f(P)$, then its modified value $f'_R(P)$ as a member of region R with maximum $M_R(x_{M_R}, y_{M_R})$ is

$$f'_R(P) = f(P) \cdot \frac{\frac{f(M_R)}{(x_P - x_{M_R})^2 + (y_P - y_{M_R})^2}}{\sum\limits_i \frac{f(M_i)}{(x_P - x_{M_i})^2 + (y_P - y_{M_i})^2}} \quad . \qquad [4.1]$$

where i loops over all adjacent regions of P. If there is only one region, the fraction equals 1 and $f'_R(P) = f(P)$.

The further analysis of the tactile image is done using moments up to the 2nd order [28]. The two-dimensional $(p+q)^{th}$ order moment $m_{p,q}$ of a region R is defined as the following double



```
       0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15
     ----------------------------------------------------------------
  0|   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  1|   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
  2|   0   0   0   0   0   0   0 136 170 124   0   0   0   0   0   0
  3|   0   0   0   0  56 142 120 171 192 174   5   0   0   0   0   0
  4|   0   0   0   0 161 180 145 121 151 130   0   0   0   0   0   0
  5|   0   0   0   0 131 149  99  13  10   0   0   0   0   0   0   0
  6|   0   0   0   0   9  42  56  69  31   0   0   0   0   0   0   0
  7|   0   0   0   0   0  67 152 170 142  12   0   0   0   0   0   0
  8|   0   0   0   0   0  59 174 191 171   7   0   0   0   0   0   0
  9|   0   0   0   0   0   0  81 140  67   0   0   0   0   0   0   0
 10|   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
 11|   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0   0
```

Figure 4.6: Contact regions with boundary pixels.

sum over all image pixels (x, y) of the region and their values $f'(x, y)$:

$$m'_{p,q} = \sum_x \sum_y x^p y^q f'_R(x,y) \qquad \text{where } (x,y) \in R \text{ and } p,q \geq 0 \ . \tag{4.2}$$

The moment $m_{0,0}$ constitutes the total area of the region. The centroid $C(x_C, y_C)$ of a region can be computed to

$$x_C = \frac{m'_{1,0}}{m'_{0,0}} \tag{4.3}$$

$$y_C = \frac{m'_{0,1}}{m'_{0,0}} \ . \tag{4.4}$$

The centroid allows to calculate the higher order moments with respect to it, the so-called *central moments* $\mu_{p,q}$:

$$\mu_{p,q} = \sum_x \sum_y (x - x_C)^p (y - y_C)^q f(x,y) \qquad p,q \geq 0 \ . \tag{4.5}$$

The 2nd order central moments

$$\mu_{2,0} = \sum_x \sum_y (x - x_C)^2 f(x,y) \tag{4.6}$$

$$\mu_{0,2} = \sum_x \sum_y (y - y_C)^2 f(x,y) \tag{4.7}$$

$$\mu_{1,1} = \sum_x \sum_y (x - x_C)(y - y_C) f(x,y) \tag{4.8}$$

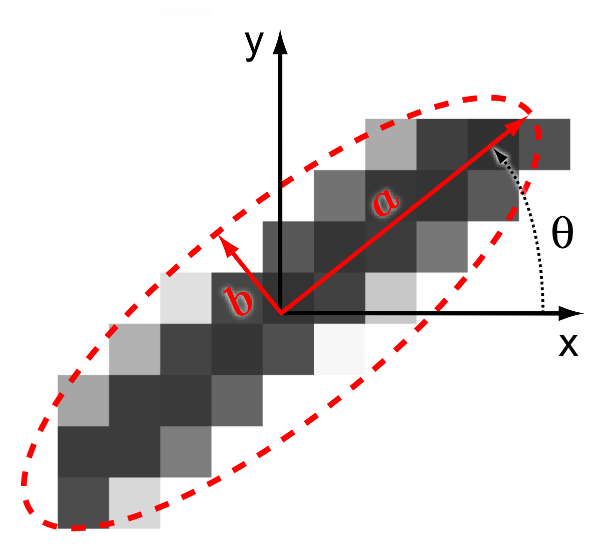


Figure 4.7: Approximation of a contact area by an ellipse.

approximate a region by an ellipse and represent its principal axes (cf. Fig. 4.7). The eccentricity ε is a measure for the roundness of an ellipse—a circle has an eccentricity of zero, whereas an oblong ellipse has an eccentricity close to one. With the 2^{nd} order central moments it amounts to

$$\varepsilon = \frac{(\mu_{2,0} - \mu_{0,2})^2 + 4\mu_{1,1}^2}{(\mu_{2,0} + \mu_{0,2})^2} \qquad \varepsilon \in [0, 1[\ . \tag{4.9}$$

With the lengths a und b of the principal axes as depicted in Fig. 4.7 there is another way of obtaining ε:

$$\varepsilon = \frac{\sqrt{a^2 - b^2}}{a} \qquad a \geq b \ . \tag{4.10}$$

Since the area A of an ellipse is

$$\begin{aligned} A &= \pi ab && [4.11] \\ &= \pi a^2 \sqrt{1 - \varepsilon^2} \ , && [4.12] \end{aligned}$$

we have a way of determining the length a, assuming an elliptic shape of the region and therefore $A = m_{0,0}$:

$$a = \sqrt{\frac{m_{0,0}}{\pi \sqrt{1 - \varepsilon^2}}} \ . \tag{4.13}$$

The length a gives us an estimate for the absolute size of a contact point as the distance between its tip and its end; this is used, e.g., to recognize the `'stop'` command as shown in Fig. 4.10(f) on p. 53. The angle θ between the principal axes and the sensor coordinate system that is shown in Fig. 4.7 can be readily calculated by

$$\theta = \frac{1}{2}\arctan\frac{2\mu_{1,1}}{\mu_{2,0}-\mu_{0,2}} \quad , \qquad [4.14]$$

although it is currently not used in the tactile language system.

4.2.4 Robots

Figure 3.1 on p. 28 shows the robot we are currently working with. You can see the black patches on the arm which constitute its artificial skin; these tactile sensors are of the same kind as the *DSAMOD-6*. The idea is to use our tactile language on those sensors directly as soon as a suitable new version of them with a low-friction surface is available.

The next robot to use the tactile language with was already depicted in Fig. 1.2 on p. 2. On this robot the tactile sensor patches are the blue patches on the shoulder and the arms.

4.3 Input Modes

There are four possible input modes when using the tactile sensor. The first mode is used to enter letters and symbols. In this mode, the contact path data is only transferred to the language interpretation system when there is no more contact detected, i.e., the complete symbol has been entered. This is the *default mode*. In the second mode, the relative coordinates of each contact point are immediately transmitted to the robot control. The origin of the coordinate system is placed at the position of the first contact. This *direct control mode* is used for the direct control of the robot's extremities. The *indirect control mode* is similar to the previous mode except that the input data is only processed after the entry is complete. The last mode, the *button mode*, divides the tactile skin into different areas that are interpreted like push buttons, analogous to a touch screen. If one touches one area the identifier of that area is returned. These input modes will now be explained in more detail.

4.3.1 Abstract Commands

The abstract command mode is for the entry of both handwritten standard alphabet characters and custom-designed symbols. The former are recognized by the freely available program *JMerlin* that we integrated into our system; the latter are recognized by a our own algorithm. Both recognition modules are described in the following.

Handwritten character recognition software *JMerlin*

JMerlin [24] was developed by Stefan Hellkvist [25]. It is an optical character recognition tool that processes singly entered handwritten characters. As a consequence, it is best suitable for

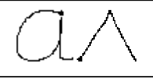
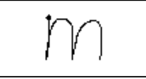
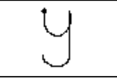
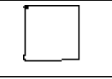
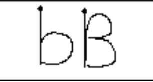
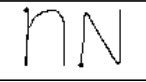
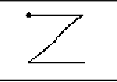
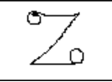
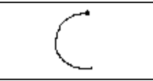
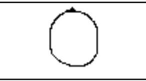
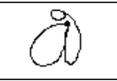
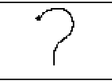
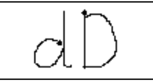
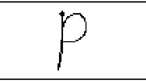
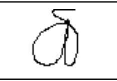
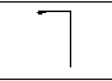
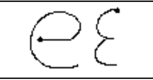
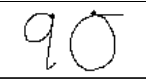
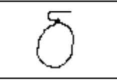
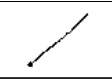
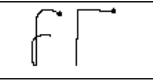
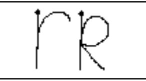
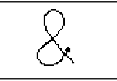
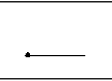
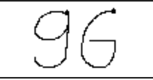
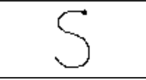
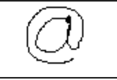
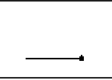
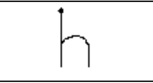
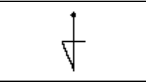
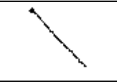
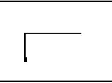
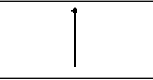
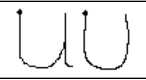
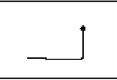
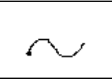
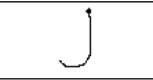
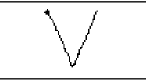
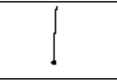
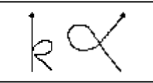
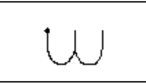
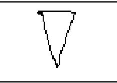
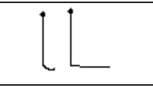
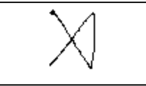
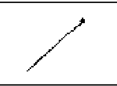
Character	Pattern	Character	Pattern	Character	Pattern	Character	Pattern
a		m		y		\0	
b		n		z		%	
c		o		å		?	
d		p		ä		’	
e		q		ö		/	
f		r		&		SPACE	
g		s		@		\b	
h		t		\		\t	
i		u		,		~	
j		v		ESC			
k		w		!			
l		x		\n			

Figure 4.8: Stroke alphabet for *JMerlin* [25].

mobile devices with a small character entry area. As input, *JMerlin*'s Java class for character recognition expects a dynamic list of type `Vector` that contains all consecutive tactile input points of the character in chronological order. The generation of these points is dependent on the tactile sensor hardware and needs to be modified to fit *JMerlin*'s needs. *JMerlin* recognizes the characters using the “elastic matching” method [47]. It is a robust technique that calculates the distance between the unknown input symbol and models of all recognizable characters to determine the best matching character. The recognition results are pretty good (see Table 4.2), but for a good recognition result it is necessary to use the stroke alphabet as shown in Fig. 4.8. The output of *JMerlin* is connected to the MCA2 framework via a TCP/IP socket.

Custom symbol recognition

The ease of use and integration of *JMerlin* lead us to expand its functionality, based on the existing data format, by a symbol recognition of multi-finger contacts. Therefore, besides of the handwritten alphabet symbols, there are several custom-designed symbols that we are able to recognize. Figures 4.9 and 4.10 give an overview over the most important ones of them and their meaning.

The symbol recognition was realized with a set of basic rules and heuristic parameters. The heuristic is used when an optimal solution is very hard or impossible to calculate. In these cases heuristics are able to provide good results with relatively little computational effort. In order to use these rules each contact path needs to be classified into the categories *line segment*, *right turn*, *left turn*, or *sigmoid turn*. The next two paragraphs explain how this is done.

Line segment recognition For the purpose of line segment recognition it is assumed that the start and end points of the paths are the beginning and the end of the line segment, respectively (Fig. 4.11). In order to test if the path is actually a line segment or not, each point of the contact path must be within a maximum distance of the line segment. This maximum distance d depends on the Euclidean distance of the start and the end point of the path:

$$d = \frac{\lambda}{k} \ , \qquad [4.15]$$

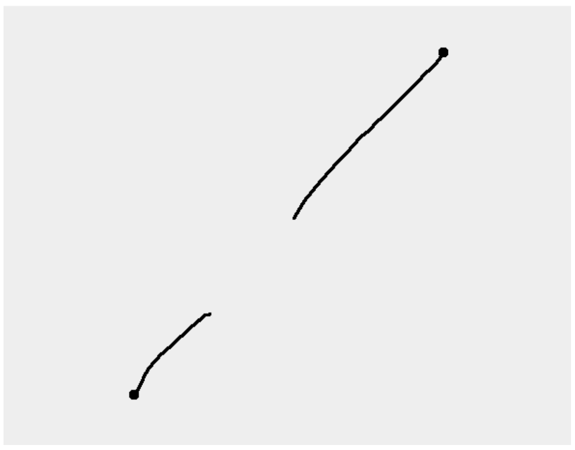
where k is a heuristic parameter that can be chosen freely.

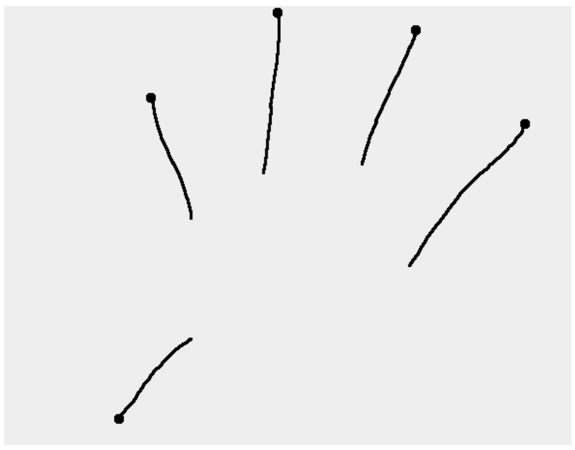
Turn recognition Due to sporadic noise of tactile sensors and the likely misinterpretation of slight disturbances as turns, we had to reduce the contact paths to a minimum of necessary information for turn recognition. Instead of using a cubic spline interpolation we opted for a simple linear approximation algorithm [60] that is depicted in Fig. 4.12. The algorithm divides the contact path into several segments. It starts with the largest possible segment that connects the start and the end point. Then it proceeds to split this segment into two smaller segments at the point of maximum distance d_{max} between path and segment if this distance exceeds a predefined threshold. This turn recognition is executed if a contact path wasn't classified as a line segment.

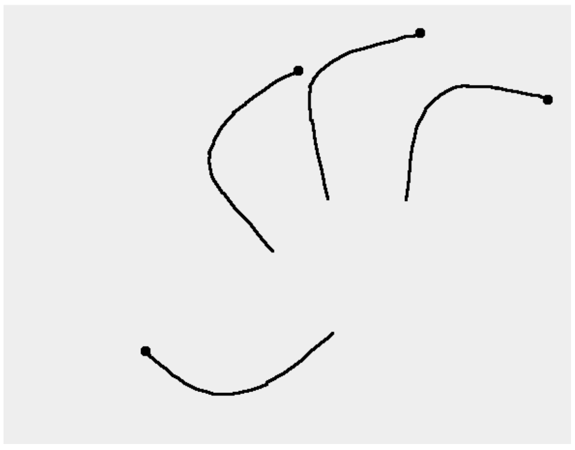
An advantage of this kind of approximation is its easy and universal applicability which, for example, is not the case for spline interpolation. Spline interpolation demands to approximate the contact path through a function f, where $y = f(x)$ must be unique. In general, it is possible to use the linear approximation for arbitrary contact paths since the given points of the path are processed in chronological order.

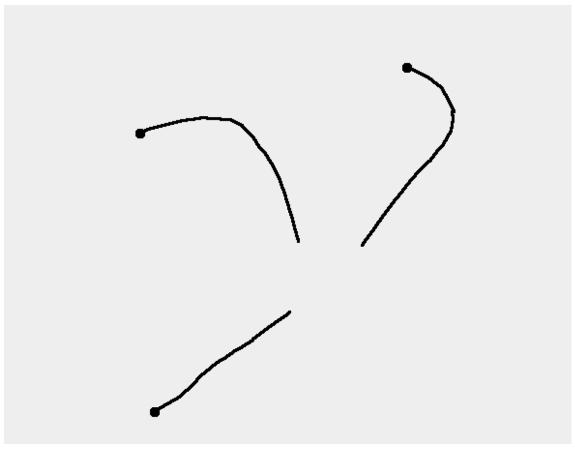
By means of this approximation we end up with a description of the input that has been reduced to a minimum of control points, with all insignificant information removed that could impede a classification.

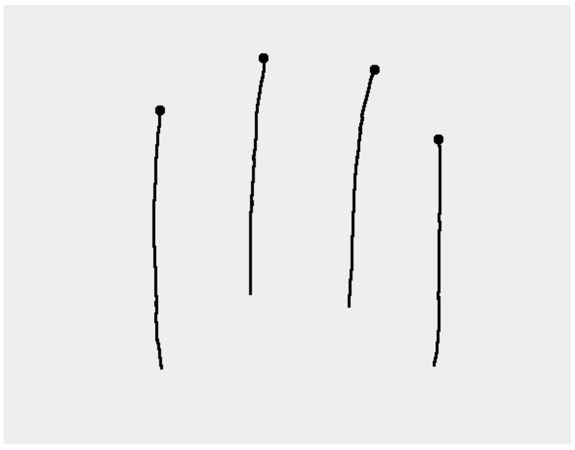
The list of control points is used to determine the type of turn. For all n the vector $\overrightarrow{v}_i = \overrightarrow{p}_{i+1} - \overrightarrow{p}_i$ is compared with the vector $\overrightarrow{v}_{i+1} = \overrightarrow{p}_{i+2} - \overrightarrow{p}_{i+1}$. For example, if vector $\overrightarrow{v}_{i+1}$ points to the left of vector $\overrightarrow{v}_i$ for all i we have a left turn. With analogous algorithms right turns and


(a) Prismatic grasp with 2 fingers.


(b) Prismatic grasp with 5 fingers.


(c) Turn back of hand upwards and Prismatic grasp.


(d) Turn back of hand downwards and Prismatic grasp.


(e) Hook grasp.

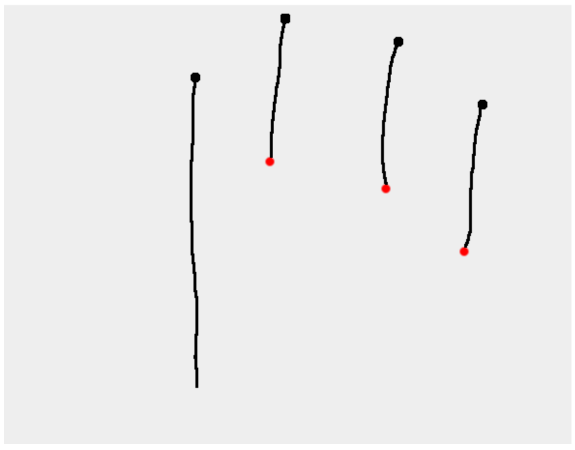
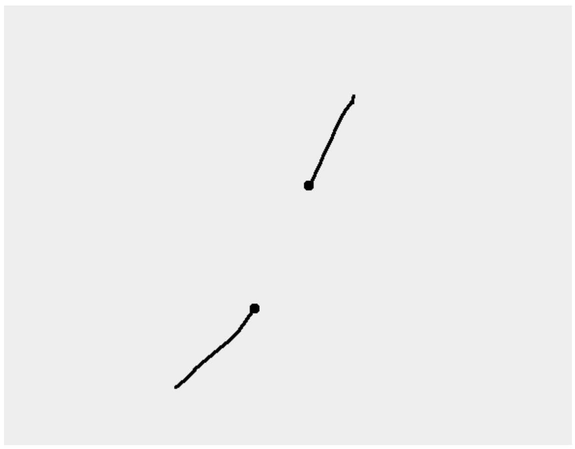
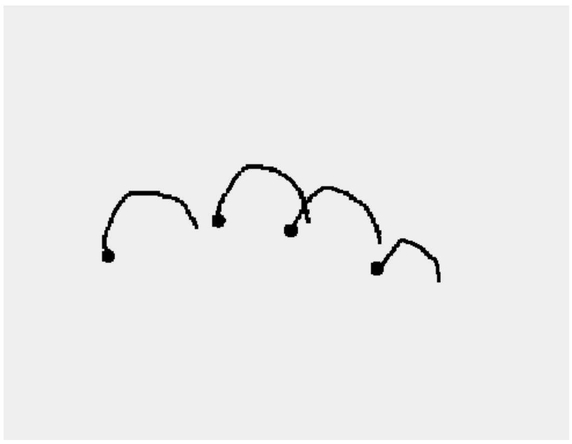
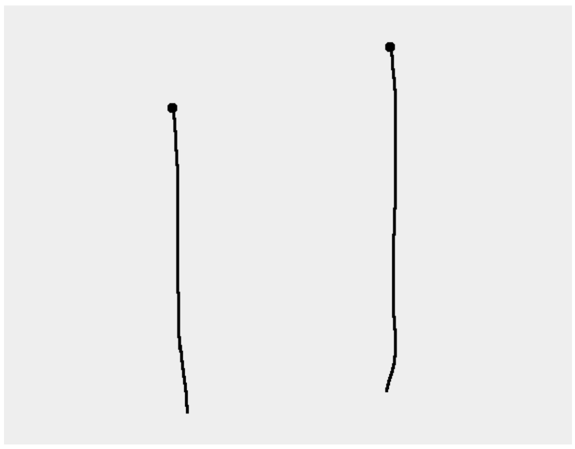
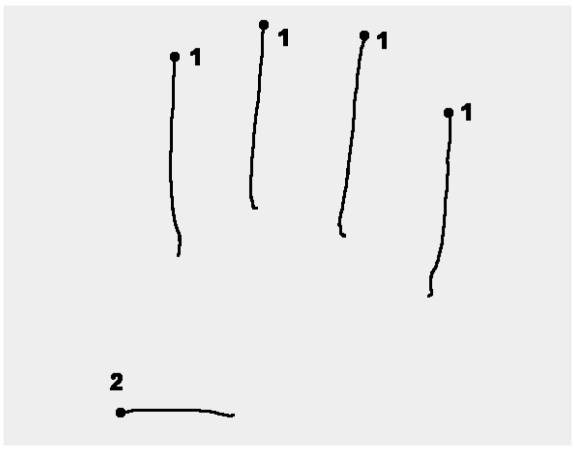

(f) Pointing index finger gesture.

Figure 4.9: Different symbols and their associated commands (part 1).

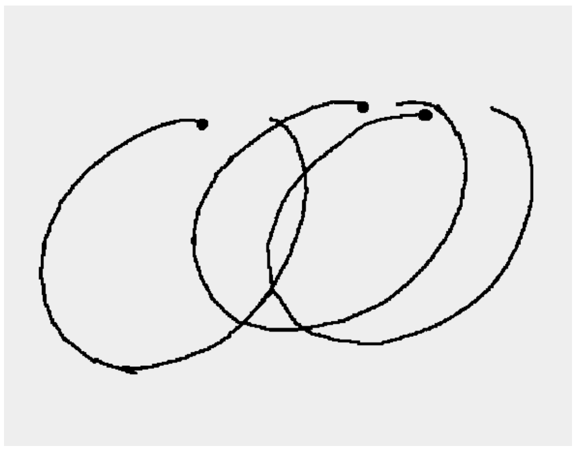

(a) Release grasp.

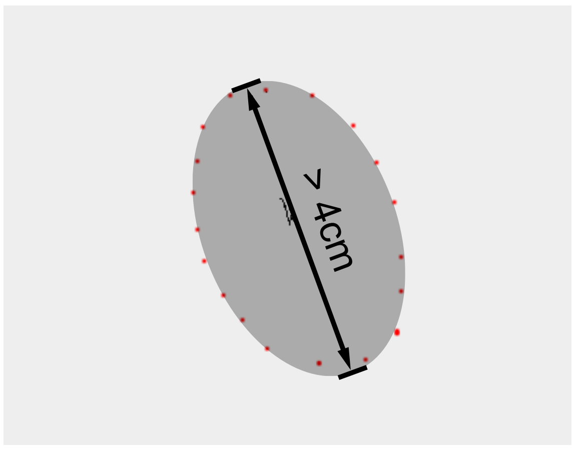

(b) Turn back of hand downwards.


(c) Home gesture.



(d) Tablet grasp.


(e) Symbol for `'object'`.



(f) `'stop'` command.

Figure 4.10: Different symbols and their associated commands (part 2).

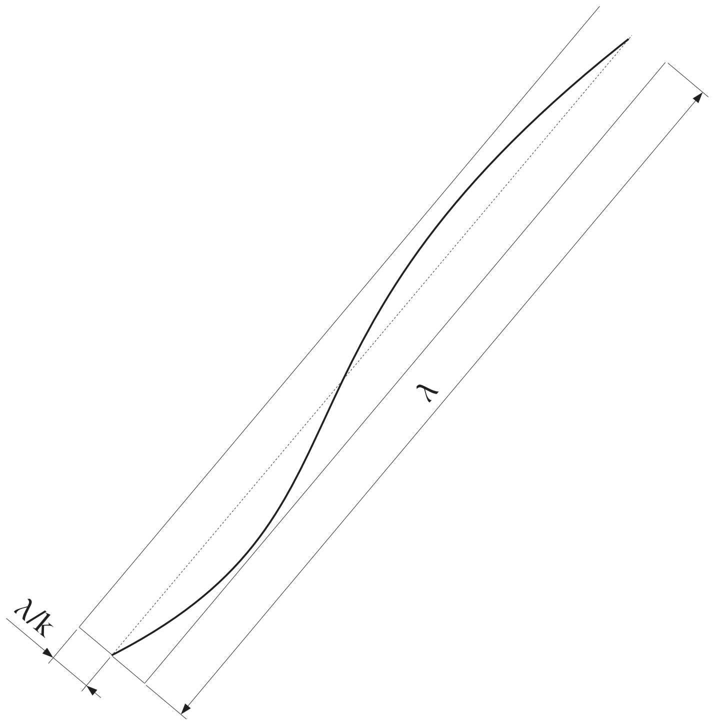


Figure 4.11: Line segment recognition.

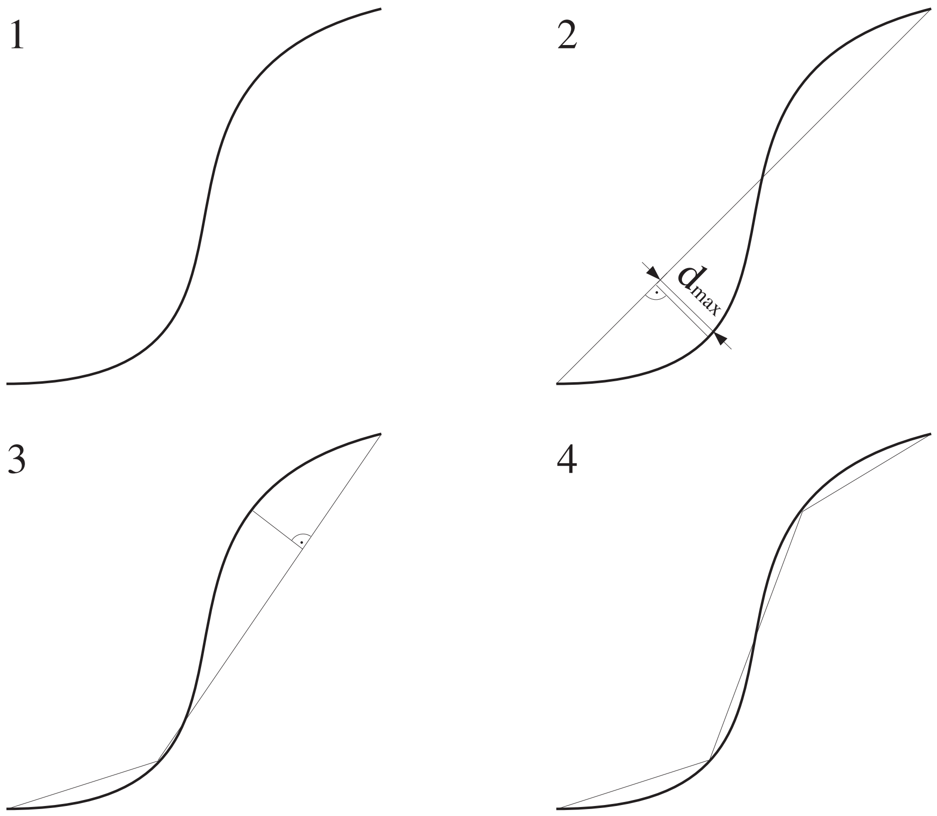


Figure 4.12: Turn recognition.

`isLineSegment(path, factor)`
`isCurve(path, d)`
`allAtSameTime(paths, t)`
`getSameTimePaths(paths,t)`
`getSameStartTimePaths(paths,t)`
`getSameEndTimePaths(paths,t)`
`getRelativeDirection(path1, path2, val)`
`getNumberOfOneDirection(paths, i, val)`
`isSmaller(paths,val)`
`isBigger(paths,val)`
`isOnLine(paths,val)`
`getPattern(path)`

Table 4.1: Functions used for symbol classification.

the two possible sigmoid turns can be differentiated.

Symbol classification As mentioned before, the actual symbol classification uses a set of simple rules that draw on basic properties of the contact paths that make up the symbol. These properties are determined by a set of functions that is shown in Table 4.1. As an example, the basic rules for the recognition of the symbol *Rotate Up + Prismatic4* (Fig. 4.9(c)) are (a) all paths are curves, (b) the contacts must be closer together at the end than at the beginning (grasp), (c) all of them are left turns (rotation), and (d) there are 4 simultaneous contacts. In pseudocode this looks like:

```
if (All paths == isCurve)
    if (All paths == isSmaller)
        if (All paths == LeftTurn)
            switch (# paths)
                case 4: Symbol = ROTUPPRISMATIC4;
```

In the interest of being able to fine-tune the algorithm, most of the functions in Table 4.1 include an additional parameter whose initial value has been determined heuristically. This is necessary to accommodate different tactile sensors as well as different users. Depending on the sensor area and resolution, the symbols are written larger or smaller, and relatively coarser or more accurate. Additionally, users tend to have different motor skills and patterns that affect their writing habits or, more generally speaking, their finger and hand movements. By adjusting these parameters, the recognition can be improved significantly. Right now, the adaption is done manually, but in a further version, this could be handled by a calibration process that uses some test symbols entered by the user to optimize the parameter set automatically.

Further symbol interpretation Apart from the actual symbol recognition and classification, further characteristics are extracted from the symbol input such as *velocity*, *pressure*, *direction*, and *length*. They serve as parameters for the generated robot command.

The *pressure* of a path is calculated as the mean pressure of all contacts, whereas the mean pressure of all paths becomes the pressure of the whole symbol. For grasping commands, the pressure serves as a parameter for the desired strength of the grip. High pressure corresponds to a strong grip, low pressure denotes a careful grasp for fragile objects. During direct robot control, the pressure signifies how strongly the robot should push against a resistance measured by the force-torque sensor.

The *speed* of a path or symbol is inversely proportional to the time difference between the last and the first contact. It translates directly into the desired speed of the robot extremity to be controlled.

The Euclidean distance of the first and the last contact yields the *length* of the path in the case of a line segment. In the case of a turn the distances of the individual contact points are summed up. Analogous to speed, the length corresponds directly to the desired distance the robot part is to be moved.

Finally, the *direction* of a line with respect to the x axis gives the target direction of a straight movement, whereas the circular arc approximating a turn gives the target *angle* to be turned during a rotation.

All these parameters can be adjusted freely by scaling factors to meet the specific demands of the user. All in all they allow for a very intuitive and efficient parametrization of the robot control commands without any additional overhead—it is all coded in one hand movement.

4.3.2 Direct Robot Control

This mode allows for the immediate control of the robot's actuators. To enter this mode, the user first needs to select the robot extremity he wants to control, then he needs to choose whether to do a translation or rotation. In the current implementation, it is possible to translate and rotate the robot's tool center point (TCP), and rotate the pan-tilt unit (the neck or head of the robot) that carries the stereo camera. Either way, after the user touches the tactile sensor the current robot position will be stored and used as the origin of the robot coordinate system. Any subsequent finger movement of the user will directly translate or rotate the chosen robot limb relative to the origin. Once the user lifts his finger off the sensor, the motion will stop, and the system will wait for the next input cycle. The mode is left by entering the `'exit'` command.

There is one complication, however. The translation and rotation of the TCP happen in three-dimensional Euclidean space along the (x, y, z) axes. The tactile sensor, on the other hand, only offers two degrees of freedom (DOF). Therefore a special mechanism was devised to alleviate this shortcoming for the translation case, which is shown in Fig. 4.13. After switching into the direct robot control mode and selecting to control the TCP, it is possible to move the TCP with the middle finger in the (x, y) plane. If the index finger touches the sensor as well, the TCP also moves in direction of the z axis, in case of the ring finger touching it moves in the opposite direction. The distance between the latter finger and the middle finger determines the speed in z direction. On top of that, the thumb can adjust the redundant elbow position of our 7-DOF robot arm. As mentioned before, this input cycle ends when the middle finger leaves the pad, and can be restarted by touching down again.

As to the rotation of TCP and robot head, the interface behaves in a similar but simpler way in that a user finger movement in the (x, y) plane triggers a corresponding motion along the pan and tilt angles of the respective limb. Thus, it is a single finger interaction. To change the roll angle of the TCP, an extra indirect control command was created (see next subsection).

4.3.3 Indirect Robot Control

In the indirect control mode, it is possible to move the chosen robot part by entering vectors for translations and arcs for rotations. Both the vector and the arc properties are extracted from every input; how the input in interpreted depends on the chosen context. The main difference to the direct control mode is that now the user input is only evaluated after the contact path has been entered completely, that is the touching finger has been lifted from the tactile sensor. The calculation of the vector length and direction is easy (Fig. 4.14(a)). For this purpose, only the first input point S and the last input point E are considered—their Euclidean distance yields the vector length, the angle between the line segment from first to last point and the y axis yields the vector direction. If an indirect translation command was chosen, the robot is moved proportional to the vector length in the (x, y) plane in direction of the vector.

The extraction of the desired rotation angle is a little more involved (cf. Fig. 4.14(b)). To that end, the input (the red stroke) is interpreted as an arc, that is as a part of the circumference of a circle (drawn in black). The desired angle α is then the central angle SCE formed by the first input point S, the last input point E, and the center of the circle C as vertex. As a consequence, the center and the radius of this circle need to be computed. The well-known equation of a circle is

$$(x - x_C)^2 + (y - y_C)^2 = r^2 \quad , \qquad [4.16]$$

where the coordinates of the center of the circle $C(x_C, y_C)$ and the radius r are the three un-

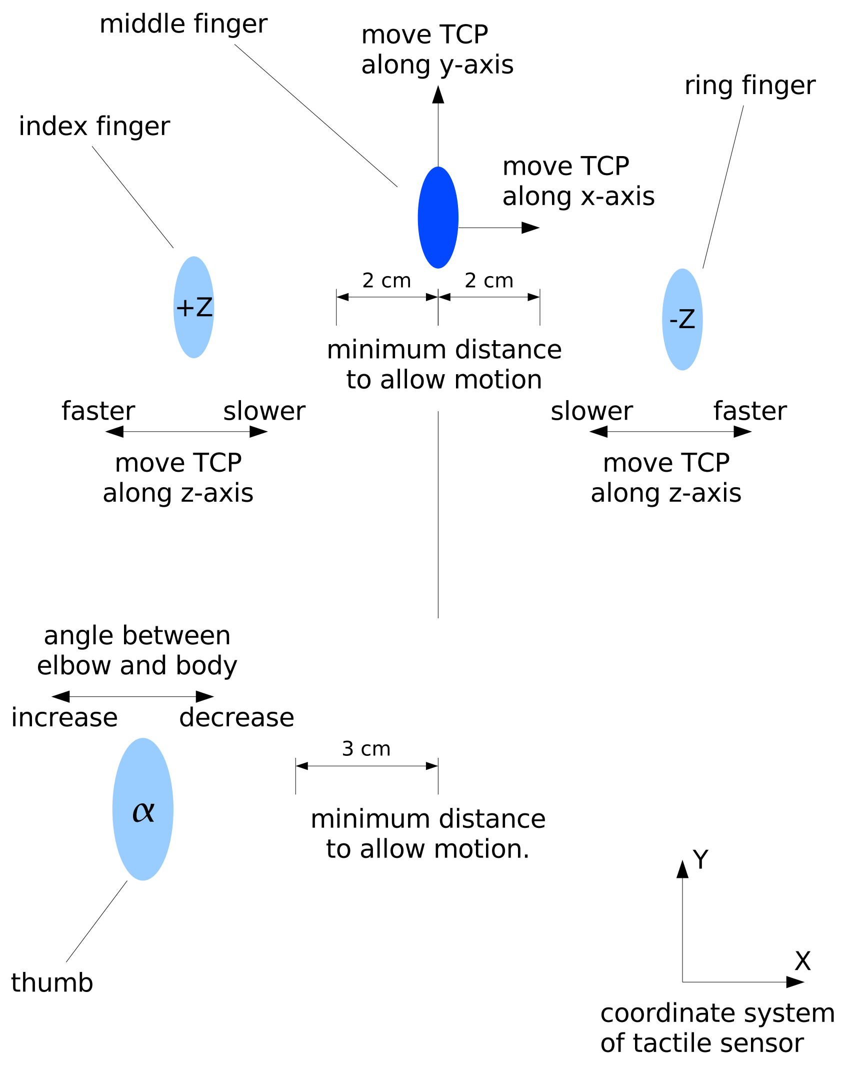


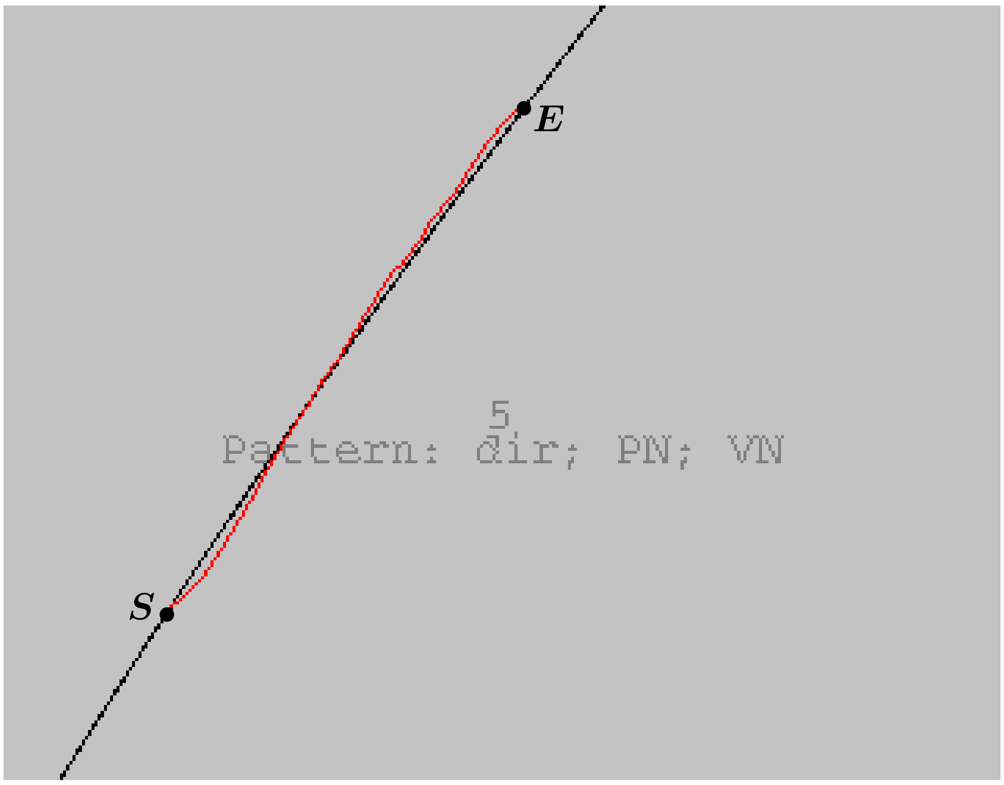
Figure 4.13: Direct control of TCP and elbow.

knowns. Hence, we need three points on the circle circumference to determine them; we opted to use the points S and E as well as that point on the contact path whose distances to the former two match at the best. The solution of this system of linear equations then leads to $C(x_C, y_C)$ and α.

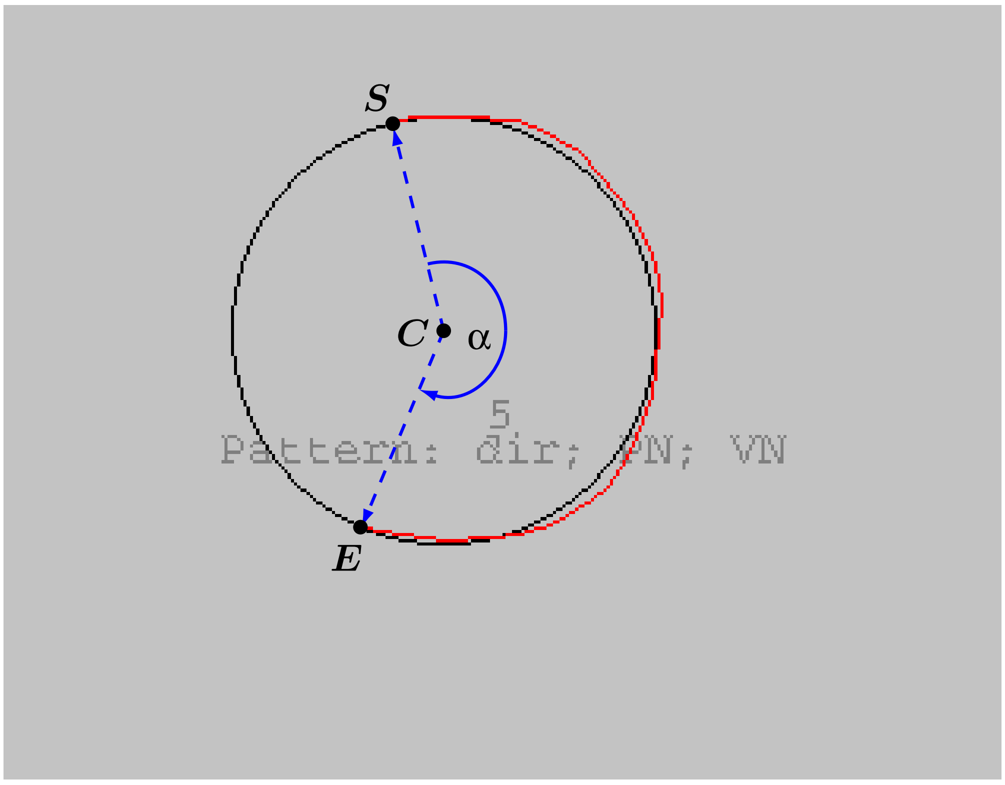
4.3.4 Button Function

In the button mode the tactile sensor is divided up into different regions that work as buttons, thus the sensor operates like a touchscreen. A button gets selected if there is only one contact path present on the sensor, and the last contact point of the path caused by lifting the finger off the sensor happens to be in a button region.

The buttons can be defined differently for each state. Currently, there are always four buttons present in the four corners per default; they can be selected anytime during the usage of the



(a) Vector.



(b) Arc.

Figure 4.14: Indirect control mode for the input of vectors (translation) and arcs (rotation).

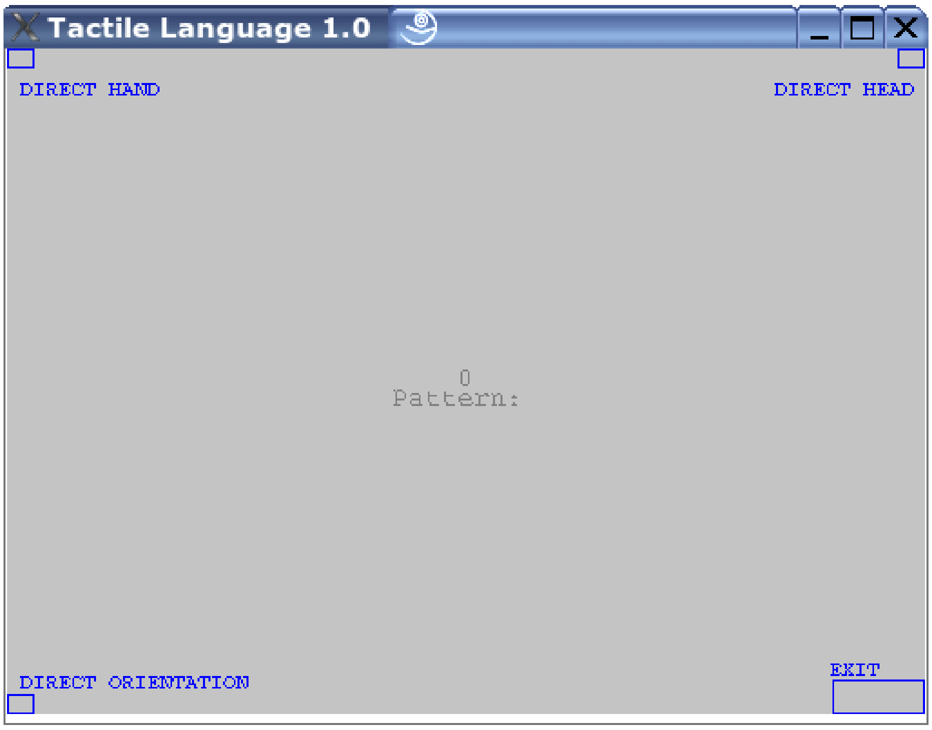


Figure 4.15: Default buttons at start state.

interface. They are shown in the screenshot in Fig. 4.15. The buttons allow for a quick selection of a robot command, or in case of the `'exit'` button the quick return to the start state.

4.4 Language Interpretation

As basic method for interpreting the language we chose a *deterministic finite automaton* (DFA). Before introducing the actual DFA design, some formal definitions and properties of DFAs will be examined [59, 27].

Definition 4.1. A DFA A is defined as a five-tuple

$$A = (S,\ \Sigma,\ \delta,\ s_0,\ F)\ ,$$

where

A	name of DFA
S	finite set of states
Σ	finite set of input symbols
$\delta : S \times \Sigma \to S$	state transition function
s_0	start or initial state, $s_0 \in S$
F	set of final or accepting states, $F \subseteq S$

The DFA is called *deterministic* if there is one and only one transition $\delta(s,\sigma)$ for every combination of a state $s \in S$ and an input symbol $\sigma \in \Sigma$. It is *finite* if there is only a finite number of states, transitions, and input symbols. It is *complete* if every state is defined for all inputs.

Apart from the formal definition 4.1, a DFA can also be represented by a state transition diagram where the states S are nodes, and the transition functions $\delta(s,\sigma) = t$ are arcs from one node s to another node t marked with the input symbol σ as the trigger for the transition.

Since we call our interface a tactile language, we are interested in the language that is recognized by the DFA. To that end, we first need to define an *extended transition function* $\hat{\delta}$:

Definition 4.2. If ϵ denotes an empty input, $x = a_1 a_2 \ldots a_n$ a word of input symbols, and $w = xa_{n+1}$ a word composed by word x and an additional input symbol a_{n+1}, then $\hat{\delta}$ is defined by

$$\begin{aligned}\hat{\delta}(s,\epsilon) &= s\\ \hat{\delta}(s,w) &= \delta(\,\hat{\delta}(s,x),\ a_{n+1}\,)\ .\end{aligned}$$

This means that an empty input causes no state transition, and that the state transition caused by an input word w is inductively defined as the state transitions due to input word x plus a final one triggered by input symbol a_{n+1}. Using this extended transition function, we can define the language $L(A)$:

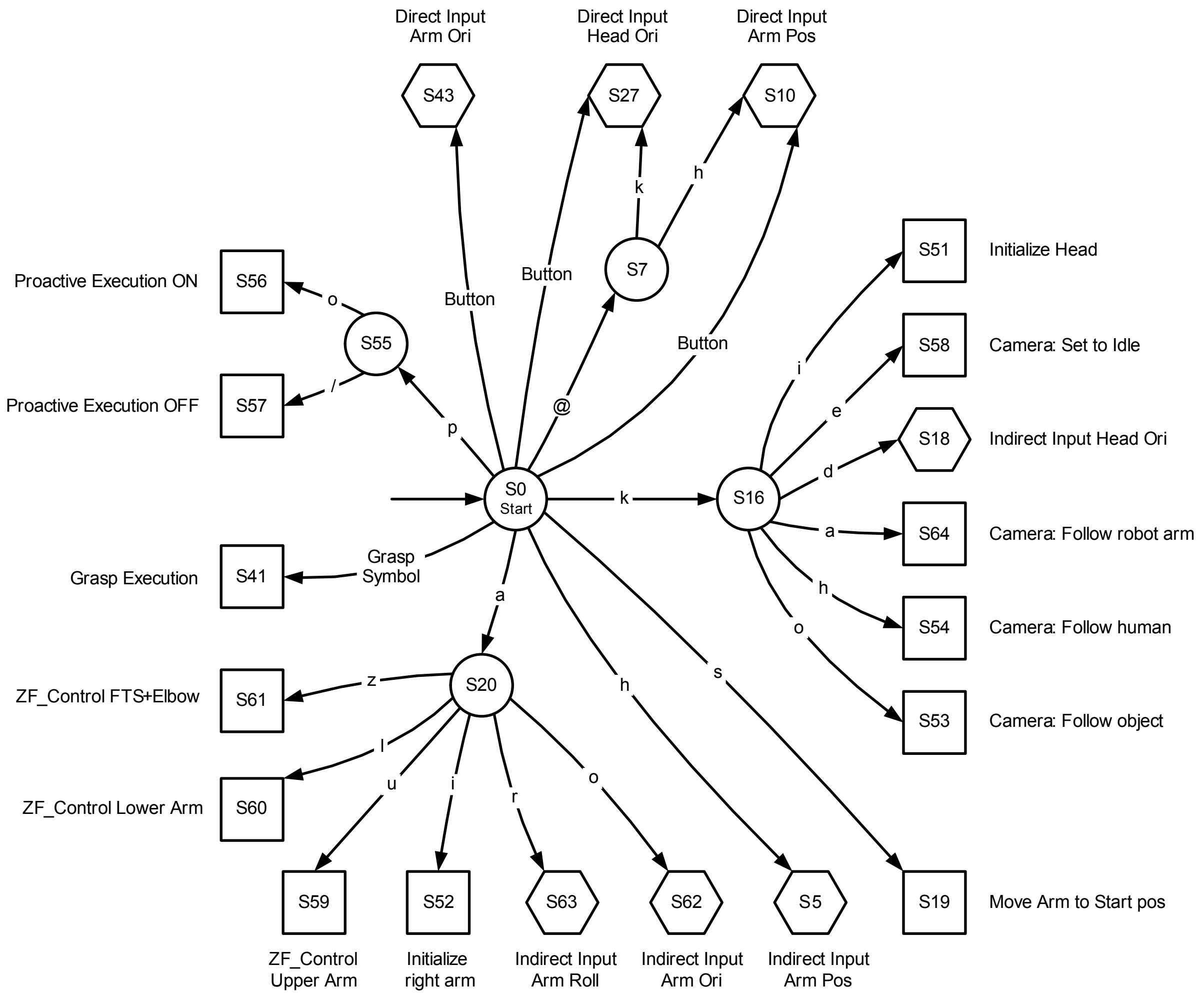
Definition 4.3. The language $L(A)$ of a DFA $A = (S,\ \Sigma,\ \delta,\ s_0,\ F)$ is defined as

$$L(A) = (w \mid \hat{\delta}(s_0,w) \in F)\ .$$

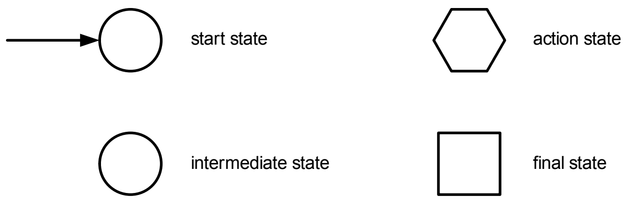
In other words: The language of A is the set of words w that transfer the current state of A from the start state s_0 to a final state $f \in F$. The language $L(A)$ of a DFA A is called a *regular language*. This reflects the fact that such a language can be generated by a *regular grammar*, which is a *Chomsky Type-3 grammar* [12].

A major part of the actual DFA we have designed is shown in Fig. 4.16. It has to be noted that our DFA is not a complete automaton, and therefore not every arbitrary word can be completely read. An erroneous entry generates an error message, but the current state is not left. For reasons of clarity, this is not illustrated in Fig. 4.16(a), but always implied. The actual mechanism is such that in case of an erroneous entry, an intermittent error state is entered from any state where a wrong input can occur. Then, the previous state is entered again. For our DFA we defined the following types of states (cf. Fig. 4.16(b)):

Start state	Entered at the very beginning of the input and after termination of an input. This is the state to start any command from.



(a) Deterministic Finite Automaton (DFA).



(b) Legend.

Figure 4.16: A fraction of the DFA we use.

Intermediate state Intermediate states merely serve as transitions to further states and therefore contain no functionality.

Action state In this state type some activity takes place at or with the robot, or a command or some information is transmitted to the robot control framework MCA2. There are also “time-dependent” states in which a command is sent to the robot if there is no further input during a freely definable time period *t*. In the other case when there actually is some valid input this “time-dependent” state corresponds to an intermediate state.

Final state A final state is an action state as well. However, after completing the activity, the state machine returns to the start state (not shown in Fig. 4.16).

As far as the implementation is concerned, a very flexible approach was adopted in that the DFA is completely specified by a text file that works along the lines of the DFA definition 4.1. A change in this file doesn’t require a recompilation of the tactile language application, it is simply read in at the beginning of the execution. An excerpt is shown below:

```
************
DFA-File 1.0
************


S:
66

SIGMA:
@,&,h,e,m,k,i,o,/,s,g,a,d,q,p,r,f,c,u,l,z,.,Prismatic2,Stop, [...]

START:
0

ACTION:
3,4,5,8,9,10,11,13,18,27,42,43,45,46,62,63

FIELD:8
e,0,8,10,5,EXIT
@h,9,0,3,3,DIRECT HAND
@k,8,2,3,3,DIRECT HEAD
@o,42,6,3,3,DIRECT ORIENTATION

FIELDACTION:
```

```
0,ALL
1,ALL
2,ALL
7,ALL

END:
2,14,15,17,19,21,23,25,26,30,32,33,34,35,36,37,38,39,40,41,44,47, [...]

DESCRIPTION:
0=The start state of the DFA
1=Indicates states for direct input.
2=Indicates the direct input until release. Next entries: e for exit, [...]
3=State for immediate hand positioning.
4=State for immediate head positioning.
[...]

DELTA:
0,&=1
0,h=5
0,@=7
0,Stop=41
[...]

1,e=2
[...]

63,dir=63
63,e=0
```

The meanings of the different fields in this file shall be explained briefly: Under S, the number of states $|S|$ is specified. Under SIGMA, the valid input symbols Σ are listed. The action states are enumerated at ACTION, whereas final states F are specified at END. The DESCRIPTION paragraph allows to put down information on the different states and is not evaluated. The DELTA section finally serves to specify the state transitions $\delta(s,\sigma)$.

We have skipped the sections FIELD and FIELDACTION in the previous explanation. They are used to define the active regions for the button mode. The first section specifies their functionality, position and label, while the second section defines the DFA states the fields are valid in.

4.5 Evaluation

In order to get a performance measure we asked 10 persons (students and staff at our institute) that were unfamiliar with our tactile language system to do several evaluation tasks. The first

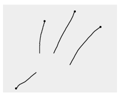
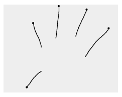
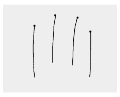
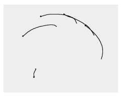
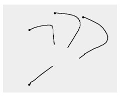
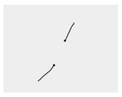
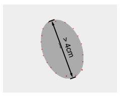
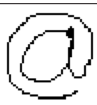
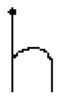
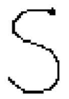
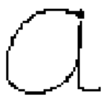
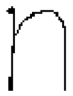
Symbol	Rec. rate [%]	Symbol	Rec. rate [%]
	87.0		81.3
	98.3		91.3
	60.0		83.0
	95.3		95.3
	89.6		95.0
	98.0		94.0

Table 4.2: Average recognition rates for selected symbols.

task tested how well some selected symbols—custom-designed symbols and handwritten alphabet letters—were recognized (section 4.5.1). For the second part of the evaluation, the test persons had to perform two more complex missions using a combination of the direct robot control and the abstract mode (section 4.5.2). Finally, the test persons filled out the NASA Task Load Index (TLX) for a subjective evaluation of this tactile user interface (section 4.5.3).

4.5.1 Symbol Recognition

For the evaluation of the symbol recognition both symbol types were evaluated. The first type consists of the custom-designed symbols that are recognized by the mechanism described above in section 4.3.1. The second type is comprised of the handwritten letters of the alphabet plus some extra characters as displayed in Fig. 4.8 on p.50. They are recognized by the JMerlin software that was integrated into the tactile language system.

For the test, the participants had to enter each symbol 40 times. In the case of the custom symbols, the users were also asked to use a specific pressure $p \in \{slight, normal, strong\}$ and velocity $v \in \{slow, normal, fast\}$. Table 4.2 shows the results per symbol averaged over all testers. It turned out that the JMerlin alphabet characters were recognized very well, with the success rates being 89.6% or higher. As to the custom-designed symbols, the results shown

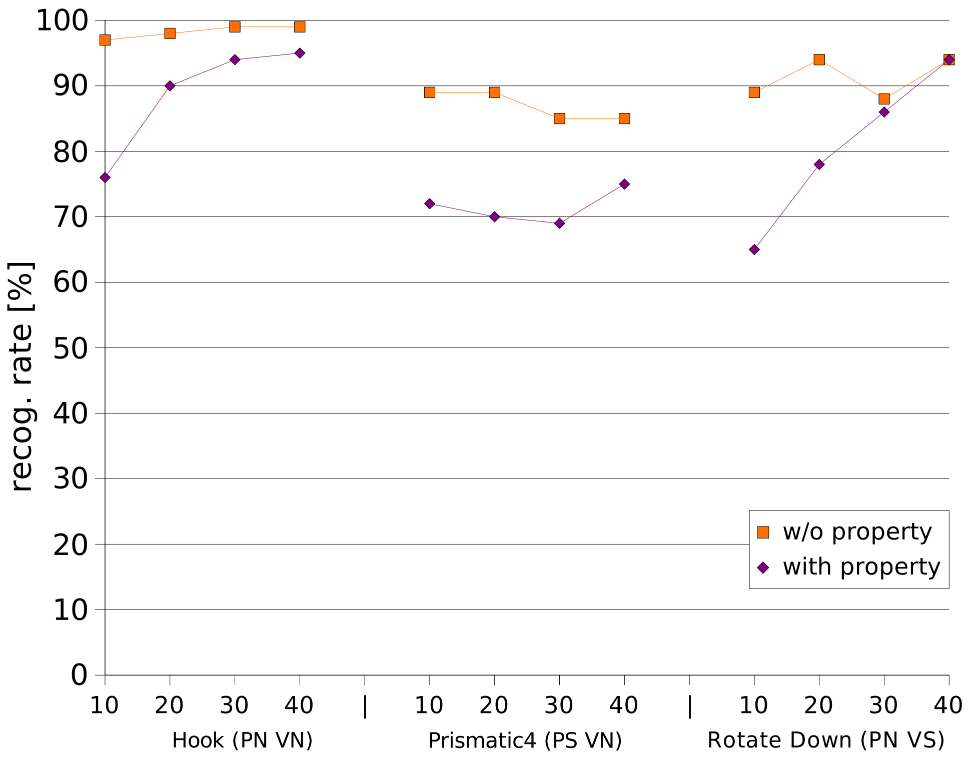


Figure 4.17: Improvement of recognition rates during test.

don't take into account the particular pressure and velocity requirements, but rather if the symbol itself was recognized correctly or not. The recognition rates were equal or greater than 81% for all of them and thus fairly good, except for the symbol *Rotate Down and Prismatic Grasp* which proved to be a more difficult symbol to enter and thus only achieved a 60% recognition rate. The most important symbol `'Stop'`, which is entered by putting or slamming the fist on the pad, had a reassuring success rate of 95.3% recognition.

To demonstrate the users' learning curves and adaptation to the system, Fig. 4.17 shows how the recognition rate improved over the course of the test; the numbers displayed represent the recognition rate over 10 consecutive symbols averaged over all users. The top curves show the rates for the recognition without regard for the pressure and velocity properties, the lower ones with them. The symbols belonging to the curves are `'hook grasp'` ($p = normal$, $velocity = normal$), `'prismatic-4finger-grasp'` ($p = slight$, $velocity = normal$), and `'rotate hand down'` ($p = normal$, $velocity = slow$). When looking at the curves, it turns out that the attribute-less recognition (a) works well, and (b) is not improving over time. With consideration of the attributes, however, there is definitely a learning curve to be seen in two of the three cases.

Since the recognition of the symbol `'Rotate hand down and Prismatic grasp'` was not satisfactory, we had a closer look at the recognition rates of custom-designed symbols of all individual test persons (Fig. 4.18). Obviously, test persons 7 and 9 had really low rates of

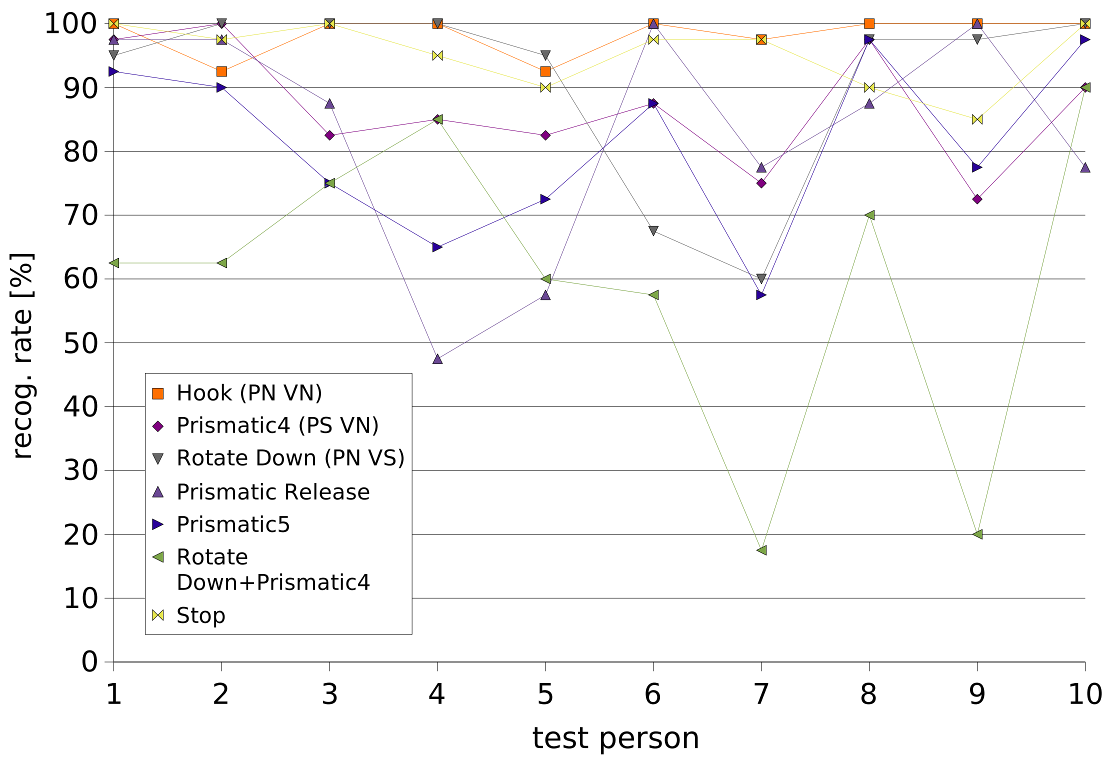


Figure 4.18: Recognition rates of the custom-designed symbols for the individual test persons.

17.5% and 20%, respectively. After analyzing their entry behavior it was found out that they either didn't close their fingers far enough or failed to touch the pad with all fingers simultaneously. Then we adapted the heuristic parameters of the symbol recognition functions that were mentioned above to better match the behavior of these testers. The result of a new recognition trial with three testers was a significant improvement in the recognition rate as shown in Fig. 4.19. The final results were 87.5% or better, even for test person 7 (unfortunately tester 9 wasn't available anymore).

4.5.2 Test Missions

For evaluating the full capabilities of this new tactile language interface, the test persons were asked to perform two more complex missions that required the usage of the direct robot control mode, the button mode, and the abstract mode. The first mission was to start from an initial robot arm position, move towards a bottle which stood about 50 cm away, grasp it with the robot hand, and finally move back to the initial position (see Fig. 4.1 on p. 42). The users thus had to switch into direct robot control mode, move and adjust the hand position along all 3 dimensions, then grasp the bottle sideways with a prismatic grasp by entering the respective symbol, and then switch back into direct control to move back to the start position. The second

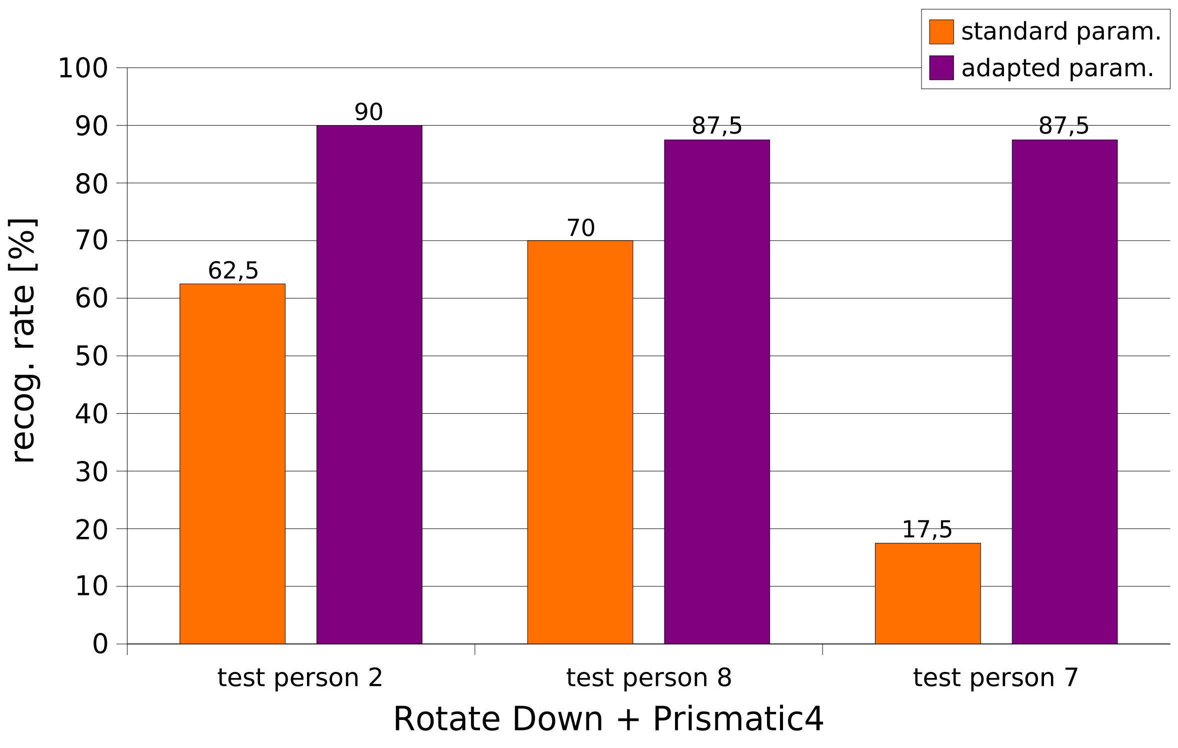


Figure 4.19: Improved recognition rates with adapted parameters.

mission consisted of starting from the same position, then moving towards and grasping a cup from the top with 2 fingers inside and the thumb on the outside. This required an additional change of the hand orientation by 90°.

Each mission was performed three times by each user. Figure 4.20 shows the mission durations for each trial, averaged over all test persons (the two top curves). As could be expected by the additional hand orientation change task, mission 2 took longer to complete than mission 1. Moreover, there is an obvious tendency of the mission durations to decrease with each additional trial (with the exception of trial 2 of mission 2), indicating that users improve their performance when using the system repeatedly. For the purpose of fathoming the remaining performance enhancement potential through repeated training, the missions were also executed by an expert user that had used the interface previously (bottom two curves). A comparison of the test person data with that of the expert user reveals that the achieved execution times were significantly lower by a factor of 2 or 3, and that even the expert was able to realize an increase in performance for these two particular tasks. Hence, there is still a lot of room for improvement for the testers concerning the usage efficiency of the interface.

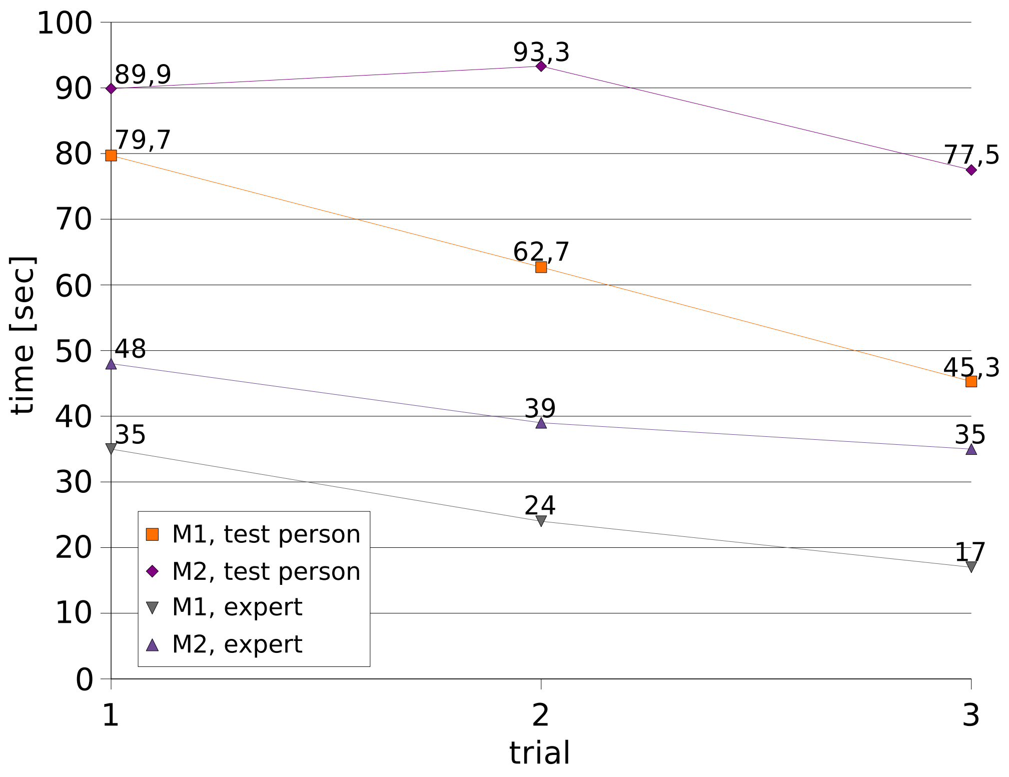


Figure 4.20: Mission completion times for each trial.

4.5.3 Subjective User Interface Evaluation

After the missions, we asked the testers to fill out the NASA TLX questionnaire [22] to report their experience with regard to six dimensions which are listed below along with their meanings:

Mental Demand	How mentally demanding was the task?
Physical Demand	How physically demanding was the task?
Temporal Demand	How hurried or rushed was the pace of the task?
Performance	How successful were you in accomplishing what you were asked to do?
Effort	How hard did you have to work to accomplish your level of performance?
Frustration	How insecure, discouraged, irritated, stressed, and annoyed were you?

Each dimension is rated on a scale from 0 to 20, where a lower score is better (meaning less stress or higher performance). The average results per dimension are depicted in Fig. 4.21.

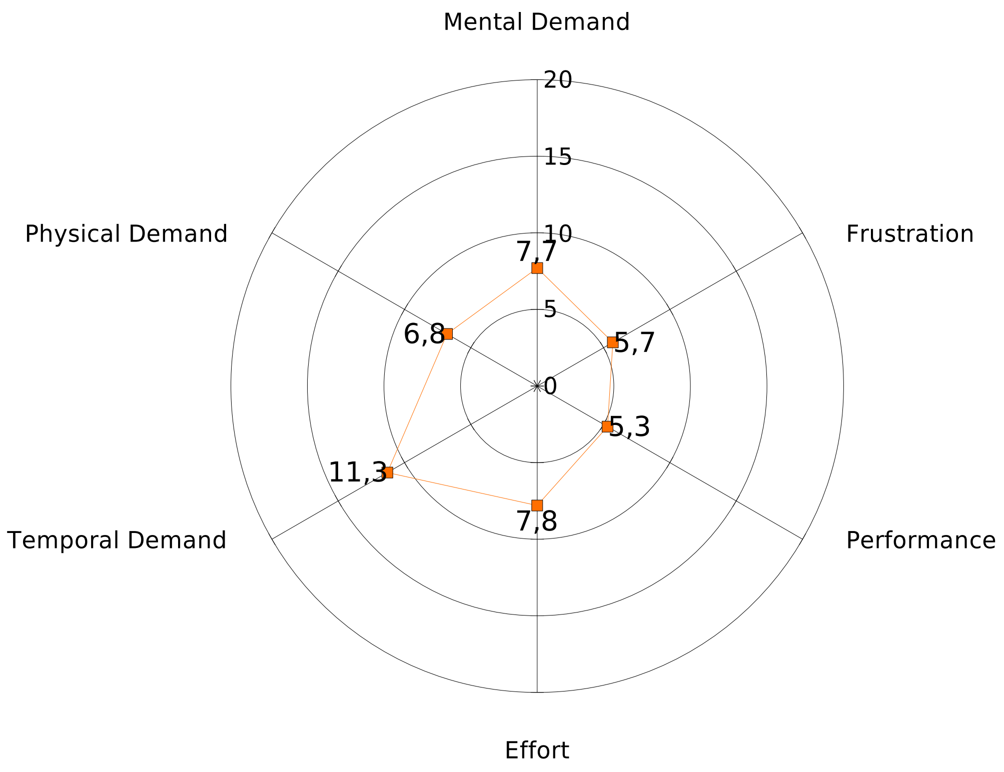


Figure 4.21: Results of the NASA Task Load Index evaluation per dimension.

Most categories were rated pretty well, namely 7.8 or better and therefore in the lower 40% of the score range. This indicates that the users were quite comfortable with the tactile interface and the results they achieved with it. However, the rating for temporal demand, i.e., the time pressure felt due to the rate or pace at which the tasks or task elements occurred, got by far the highest score. We attribute this to the testers having in mind that their performance would mainly be measured by their mission completion times, and thus causing perceived temporal stress.

4.6 Summary

This chapter demonstrated a tactile language for controlling a robot through its artificial skin. Specifically, it detailed the definition of an interface for arbitrary tactile sensors, the symbol recognition for multi-finger contacts, and the integration of a freely available character recognition software. Furthermore, the chapter outlined the interface's versatility that comes from its four different tactile input modes, the two most important being the direct control of the robot's TCP and the default mode where abstract, complex robot commands can be issued by entering appropriate multi-finger symbols. In addition, it was shown that the tactile language also very efficient since multiple parameters like direction, distance, and speed can be decoded from a single human finger stroke. Finally, the results of a user evaluation of the tactile language system were presented, namely good recognition rates of the entered symbols and a high user satisfaction.

5 Proactive Execution

Intuitive human-robot cooperation poses a challenging task to robots as it requires a high level of understanding of the human user. The approach taken here is to estimate the human intention and select and proactively execute an appropriate robot task without requiring an explicit user command. This chapter describes the concept and details of our proactive execution module and how it forms an integral part of our intuitive human-robot cooperation system. We also present implementation details and the results of an evaluation scenario.

One of the key challenges of human centered computing is the intuitive human-like interaction with robots. It requires the development of highly sophisticated approaches, since it involves the application of sensors and actuators in a real world scenario dealing with humans.

Our approach employs an intention recognition module that attempts to infer the human user's intention from the robot's sensor readings within a probabilistic framework [54]. The other module—and the focus of this chapter—is the proactive execution module that interprets the output of the intention recognition and derives a suggestion to the task planner as to what task to execute next. The overall concept of both modules was introduced in [50], while a concise version of the proactive execution module was published in [51]. The implementation of the proactive execution module is described in detail in [67].

The remainder of this chapter is structured as follows: In section 5.1 we motivate the problem. Section 5.2 outlines where in the robot architecture the proactive execution is situated. The related topic of intention recognition is introduced in Section 5.3. Section 5.4 explains how the proactive execution module works in general. The filtering steps are discussed in sections 5.5 and 5.6. The algorithm for determining the possible intentions is presented in section 5.7, while the action selection mechanism is detailed in the following section 5.8. We discuss implementation details in section 5.9 and present the results of an evaluation scenario.

5.1 Motivation

The goal of this research is intuitive human-robot cooperation in the sense of avoiding explicit clarification dialogs and explicit commands.

Humans do a good job in mutual control of their interaction by reading and interpreting the affective and social cues of each other [9]. Ergo, a robot that is able to read the user's non-verbal cues and thus infer the user's intention is able to interact more intuitively from a human perspective.

As humans try to figure out their interaction partner's goals or desires, they try to trigger reactions to get more information. A good example is a waiter on a cocktail party who wants to know if somebody wants a refill. He approaches the guest with a bottle, causing the guest to thrust out his glass or to withdraw it. We call this action of the waiter *proactive*, since he acts without an explicit command from the guest, provoking a clarifying reaction from the guest and on that account removing any uncertainty about the guest's intention. As this example illustrates, humans are accustomed to perform intuitive cooperation. Consequently, providing service robots with such a skill opens a new dimension in human-robot cooperation.

Since even humans cannot do this perfectly, recognizing the user's intention is inherently difficult. Hence a probabilistic approach has to be used. This allows for stating how certain the recognized intention is. The proactive execution module is then used to deal with and minimize this uncertainty. The challenge is to select a robot action that urges the user to react in a way that unravels the user's intention. The corresponding robot actions need to be executed with care, since the recognized intention is uncertain. The human user is meant to close the loop of intention recognition and proactive action planning.

5.2 Module Context

The context of the proactive execution module is depicted in Fig. 5.1. It receives its input from the intention recognition. This input forms the information basis on which it selects an appropriate robot action. This selected action is passed on to the task planner as the current implicit user command. In case there is no explicit user command from one of the man-machine interfaces (MMIs, on the top left branch) and the robot is idle, the implicit command will be executed.

5.3 Intention Recognition

Although not part of this research, the intention recognition mechanism shall be introduced here because the proactive execution module depends on its input; further details can be found in [54, 50].

The intention recognition model assumes that there is only one single intention that a human follows at any time (cf. Fig. 5.2). This intention causes the human to take actions. These actions in turn have an impact on the environment, and this impact is what the robot can perceive with its sensors. In addition, the current context such as the room and situation the human is in influences the human intention.

The method that was chosen to represent this described user model is a hybrid dynamic Bayesian network (HDBN). It is illustrated in Fig. 5.3. The Bayesian network nodes represent random variables, and the edges point from causes to effects. It is dynamic because the intention

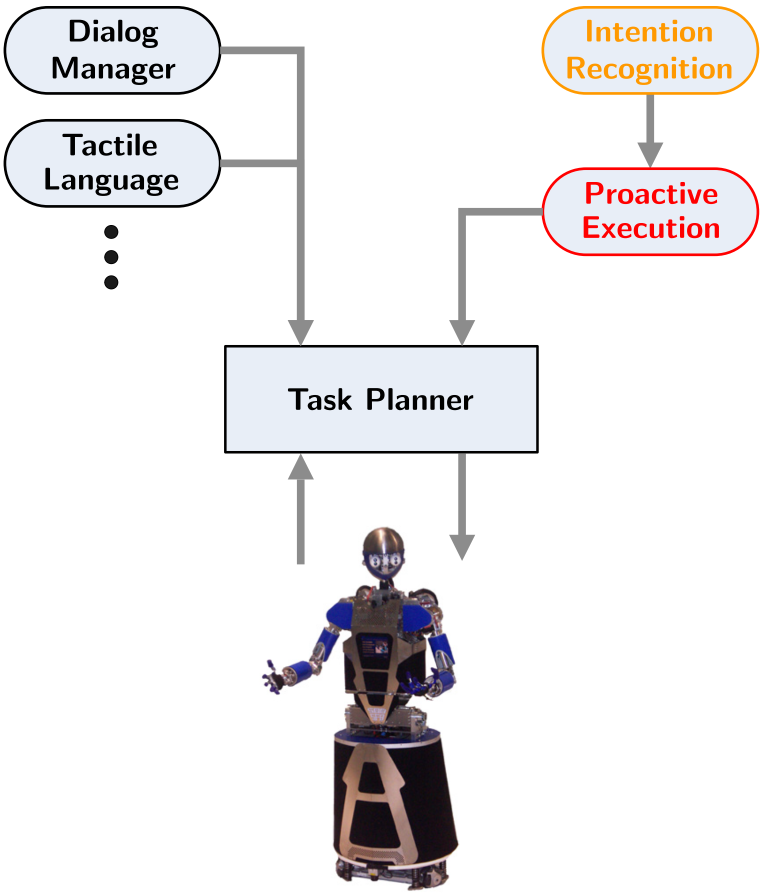


Figure 5.1: Context of the proactive execution module.

and the actions can change over time, and the model needs to reflect that accordingly. The hybrid nature of the HDBN is due to the fact that both discrete (e.g., the intention) and continuous (e.g., location of the human) information needs to be represented. For the actual estimation, the random variables are implemented as probability density functions (PDFs) generated by Gaussian mixture densities. In order to calculate the intention state of the next time step t_{n+1}, a Bayesian forward inference is performed using the current intention i_n and domain state d_n and a function $f(i_{n+1}|i_n, d_n)$ that describes the system change from one time step to the next based on these states. In addition, the sensor measurements are incorporated by calculating their probable cause through Bayesian backward inference. This first updates the action states, then in another step the intention state. On that account one finally arrives at a new PDF of the intention state at each time step which serves as the input for the proactive execution. A graphical example intention PDF is shown in Fig. 5.5(a) on p. 80.

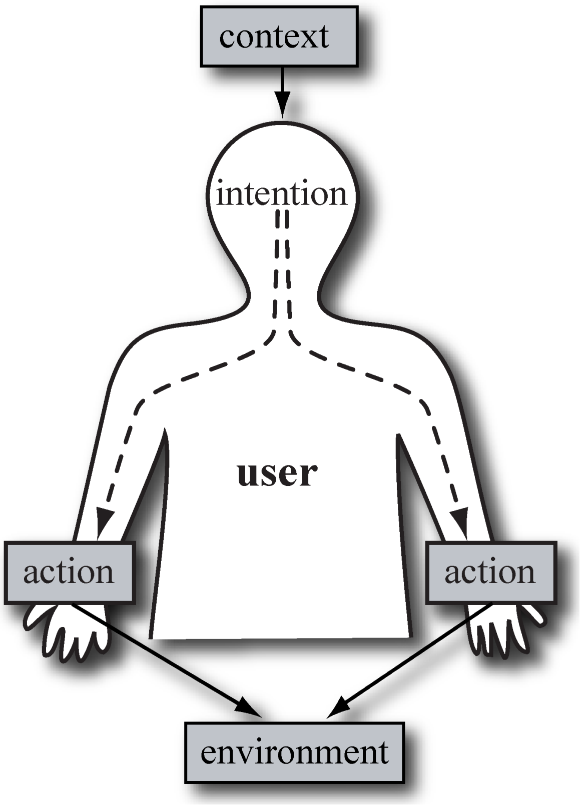


Figure 5.2: The user model for the intention recognition.

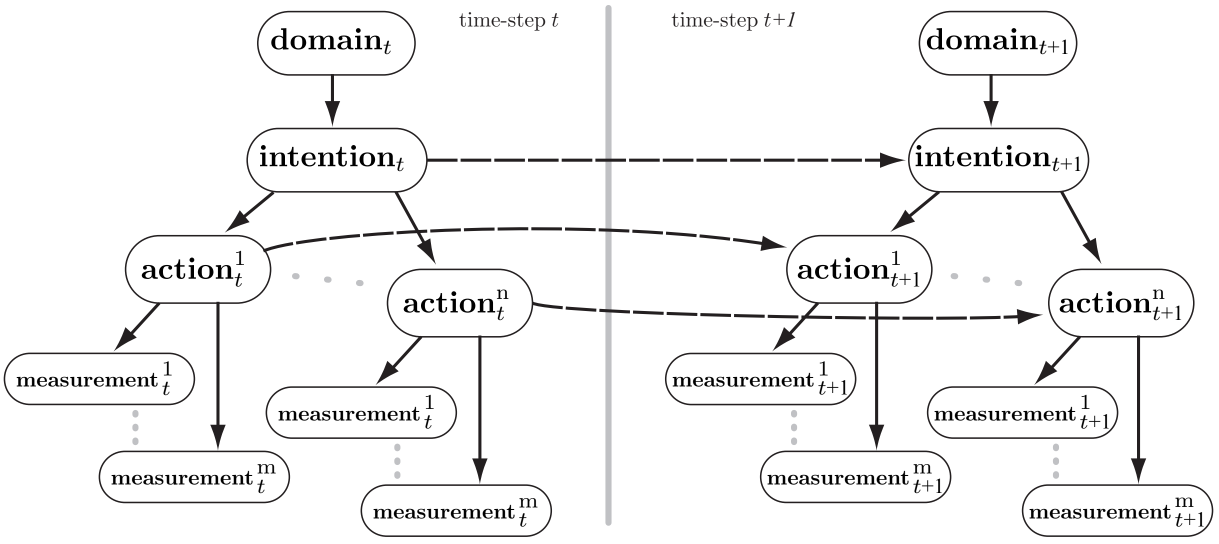


Figure 5.3: The hybrid dynamic Bayes network (HDBN) model for the intention recognition [50].

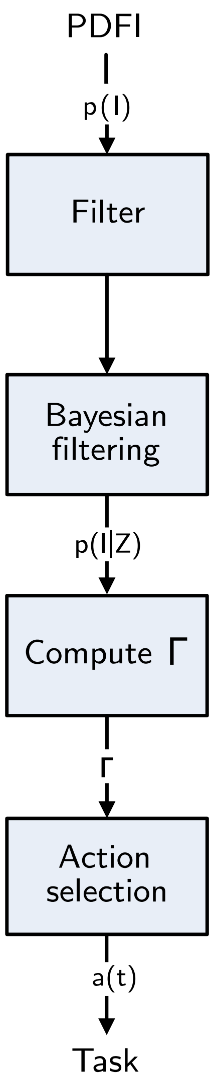


Figure 5.4: Overview over the proactive execution module workflow.

5.4 Proactive Execution Workflow Overview

Figure 5.4 shows the basic workflow of the proactive execution module. It consists of a loop that contains the following processing steps at cycle n:

- Acquire the probability density function over the human intentions (PDFI) from the intention recognition module
- Filter the PDFI
- Determine and apply the base probability of the PDFI
- Determine the set of possible intentions
- Action selection

* Execution of obvious intention
* No clear tendency
* Proactive execution (PE)

Then the next cycle $n+1$ follows. The following sections detail this process step by step.

5.5 Filtering Step

The recognition of the human intention in a real-world scenario is bound to contain uncertainty. This uncertainty is therefore modeled explicitly by a probabilistic framework that results in a probability density function over the considered human intentions (PDFI). Additionally, however, there exists also an implicit uncertainty in the process that results in changes of the PDFI from cycle to cycle that are unlikely to reflect any real changes of the intention. As a consequence, this filtering step is necessary to smooth the PDFI values over time.

5.5.1 Median Filter

To get rid of outliers we first apply a median filter. It works on a sequence of values within a window of size N, where N is chosen to be uneven. If we denote the sequence with x, this window w at time $t = 0$ looks like this:

$$w = \{x_{-(N-1)},\ x_{-(N-2)},\ \dots\ ,\ x_{-1},\ x_0\} \ . \qquad [5.1]$$

The median filter sorts the elements of w according to their size, then selects the middle element of that sorted list as filtered value y_0. Two consequences follow from that: The median filter is non-linear, and it introduces a latency of $(N-1)/2$. For our implementation of the median filter, we chose a window size of $N = \texttt{FILTER\_DEPTH\_MEDIAN} = 3$. This allows for the elimination of singular outliers.

5.5.2 IIR Filter

As a second smoothing operator, a 2^{nd} order infinite impulse response (IIR) low-pass filter is employed with the transfer function

$$H(z) = \frac{b_0}{1 + a_1 z^{-1} + a_2 z^{-2}} \ . \qquad [5.2]$$

The calculation of the resulting value is done using the related difference equation

$$y_n = -a_1 y_{n-1} - a_2 y_{n-2} + b_0 x_n \ , \qquad [5.3]$$

where x and y are the input and output sequences of the filter, respectively. Depending on the successful action selection of this proactive execution module, the cutoff frequency of the low-pass filter is adjusted—the more success, the higher the cutoff frequency, and thus the quicker the adaptation to changes in the PDFI. Successful action selection means selecting the task the human expects. This can be measured by the frequency of human-triggered task abortions.

5.6 Bayesian Filtering

The *base* or *a priori probability* of the PDFI, or $p(\hat{I})$, is the accumulated knowledge of all previous PDFI values. It serves as a ground truth, or world knowledge, as it were, that is combined with the current value of the PDFI to compute an *a posteriori probability* of the PDFI.

The calculation of the base probability is done by a recursive learning formula. With $P_n(\hat{i}_j)$ denoting the base probability of intention j at time step n, and $P_n(i_j)$ the j^{th} element of the PDFI at time step n, this formula is

$$P_n(\hat{i}_j) = \frac{\alpha P_n(i_j) + \beta P_{n-1}(\hat{i}_j)}{\alpha + \beta} \quad , \tag{5.4}$$

where $\frac{\alpha}{\alpha+\beta}$ constitutes the learning rate. All $P_0(\hat{i}_j)$ are initialized to $1/N$ at the beginning. This dynamic learning formula guarantees the continuous update of the a priori probability such that the robot can adapt to new surroundings and situations over time.

The *a posteriori* probability, $p_{post}(I|\hat{I})$, is calculated in analogy to Bayes' theorem, considering the current value of the PDFI, $p(I)$, the observation, and the base probability of the PDFI, $p(\hat{I}|I)$, the update rule given an observation:

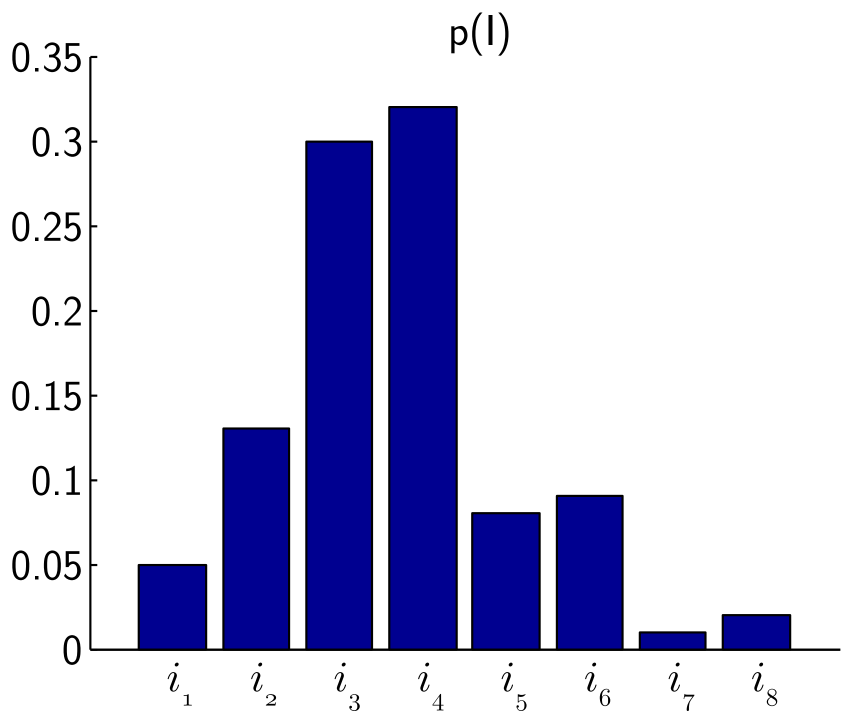
$$p_{post}(I|\hat{I}) = c \; p(\hat{I}|I) \; p(I) \quad , \tag{5.5}$$

with c being a normalizing constant that makes $p_{post}(I|\hat{I})$ sum up to 1. The N-tuple $p_{post}(I|\hat{I})$ is the PDFI version that forms the basis for all following processing steps.

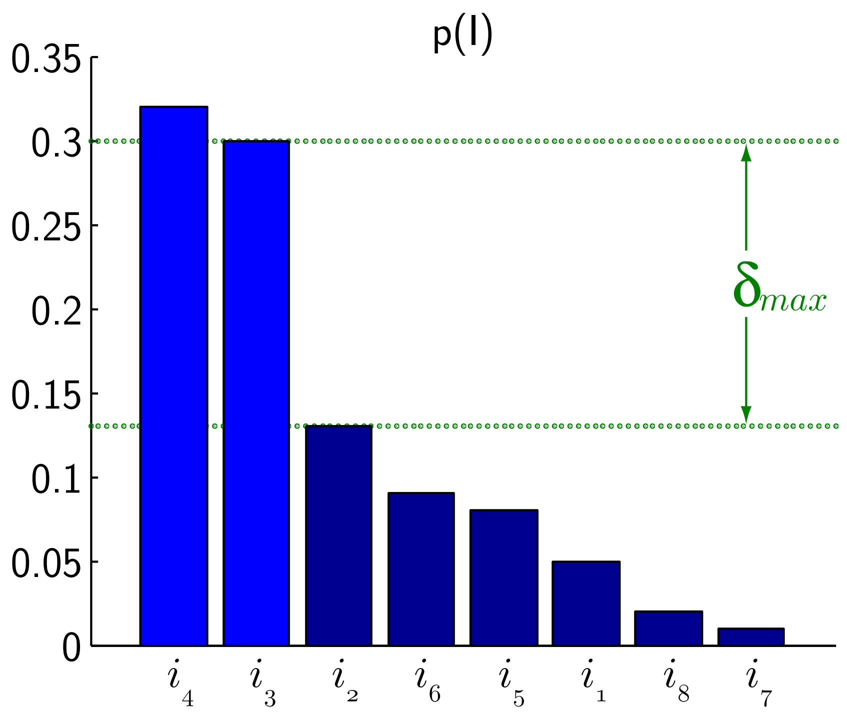
5.7 Possible Intentions

Our concept of intention recognition assumes that the human being observed by the robot has only one intention at a time (cf. [54]). Due to the inherent uncertainty reflected by the PDFI, it is unlikely that one particular intention i_j in the PDFI equals 1, and all others 0. In this processing step, those intentions that could possibly be the "true" intention are determined. They are collected in the set of *possible intentions* Γ.

The algorithm we use to determine Γ is illustrated in Fig. 5.5 and works like this:



(a) Original PDFI.



(b) PDFI after sorting and determination of δ_{max}.

Figure 5.5: Determining the set of possible intentions Γ.

	i_1	i_2	i_3	i_4	i_5	i_6
a_1	1					
a_2		1				
a_3			0.5			
a_4			0.5			
a_5				1		
a_6					1	
a_7						0.5
a_8						0.5

Table 5.1: Action selection depending on intentions $\equiv p(a_i|i_j)$.

1. Sort the intentions $p(I) = \{i_1, \dots, i_N\}$ according to ascending probabilities. This yields the list $l = \{\widetilde{i}_1, \dots, \widetilde{i}_N\}$. For each list element, the original intention identity is stored.

2. The difference of the probabilities of adjacent list elements is calculated, which yields a list $\Delta = \{\delta_1, \dots, \delta_{N-1}\}$, with $\delta_j = P(\widetilde{i}_{j+1}) - P(\widetilde{i}_j), j \in \{1, \dots, N-2\}$.

3. Determine the list position m of the maximum in Δ.

4. If the maximum difference δ_{max} is below a pre-specified threshold `THRESH_MAX_DIFF`, then $\Gamma = \{\}$; for our implementation we chose a threshold of `THRESH_MAX_DIFF` $= 1/2N$. Otherwise, the list of possible intentions becomes $\Gamma = \{\widetilde{i}_j \mid j > m,\ j \in \{1, \dots, N\}\}$.

In the example of Fig. 5.5, we therefore have $\Gamma = \{i_3, i_4\}$.

5.8 Action Selection

As the name suggests, the action selection step is responsible for choosing an appropriate robot task to execute, given the estimated human intention. The list of N recognized intentions and M possible robot actions are intrinsic properties of the robot hardware and control software, and hence are well-known in advance. In order to match the robot actions to the intentions, a human expert has to create an action selection table as shown in Table 5.1 for $N = 6$ and $M = 8$.

With such a table, it would normally be straightforward to pick an appropriate action given a certain idea of the intention. Due to our uncertainty about the intention, however, we need to distinguish several cases depending on the number of possible intentions $|\Gamma|$ and the set of thus possible actions Γ_A. In each case, each possible action is assigned a value we call *action tendency* $t(a_k)$. A graphical depiction of the different cases is shown in Fig. 5.6 and described in the following:

Case 1a If the number of possible intentions $|\Gamma| = 0$, then no intention is prevalent. All actions get an action tendency of zero.

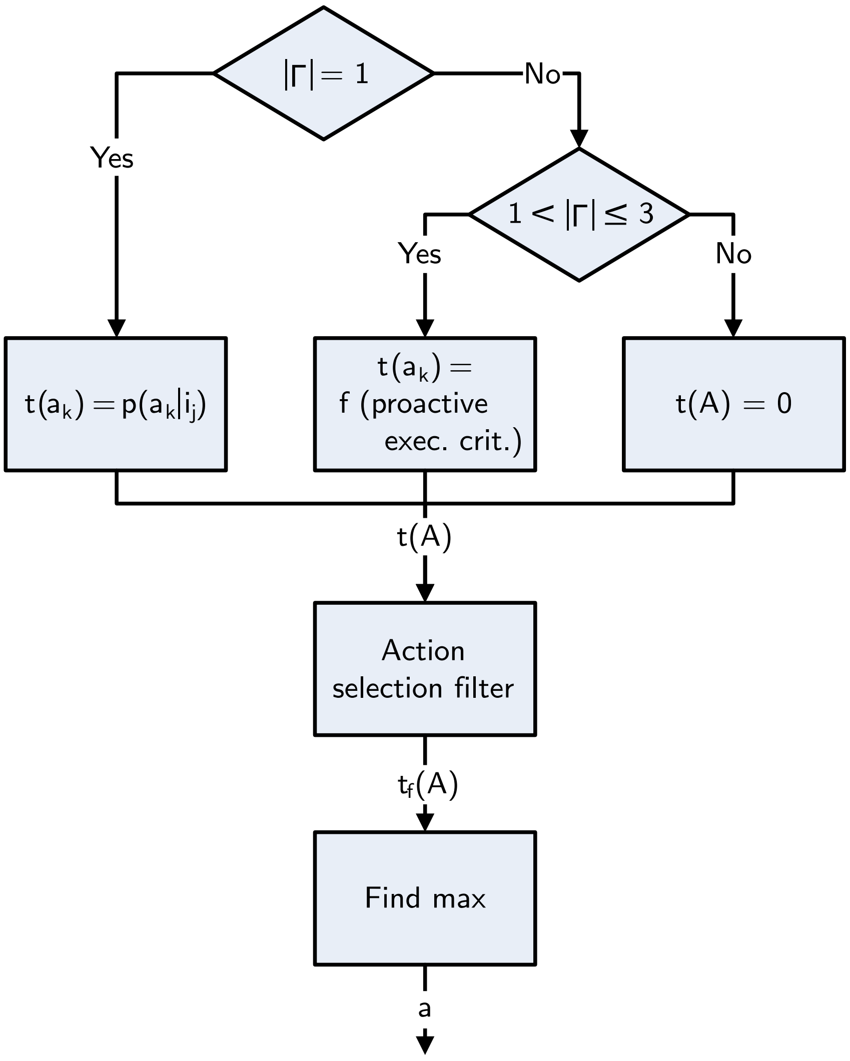


Figure 5.6: The action selection process.

Case 1b If $|\Gamma|$ is greater than a threshold `PE_MAX`, again no intention is dominating, and ergo all actions are assigned an action tendency of zero as well. We currently use `PE_MAX` = 3.

Case 2 If $|\Gamma| = 1$, there is a unique intention i_j: $\Gamma = \{i_j\}$. The actions a_k, $k = \{1, \dots, M\}$ receive their action tendencies according to the action selection table, i.e., $t(a_k) = p(a_k|i_j)$, which corresponds to the j^{th} column of the table (cf. Table 5.1).

Case 3 The ambiguous case occurs when $1 < |\Gamma| \leq$ `PE_MAX`. We call this the *proactive execution* case because the robot is not idling but will rather strive to clear up the ambiguity through his actions. In this case, the action tendencies in Γ_A are assigned by the proactive execution criteria as described below in subsections 5.8.1 to 5.8.6.

At the end of the action selection process, the action tendencies are filtered. The filter is graph-

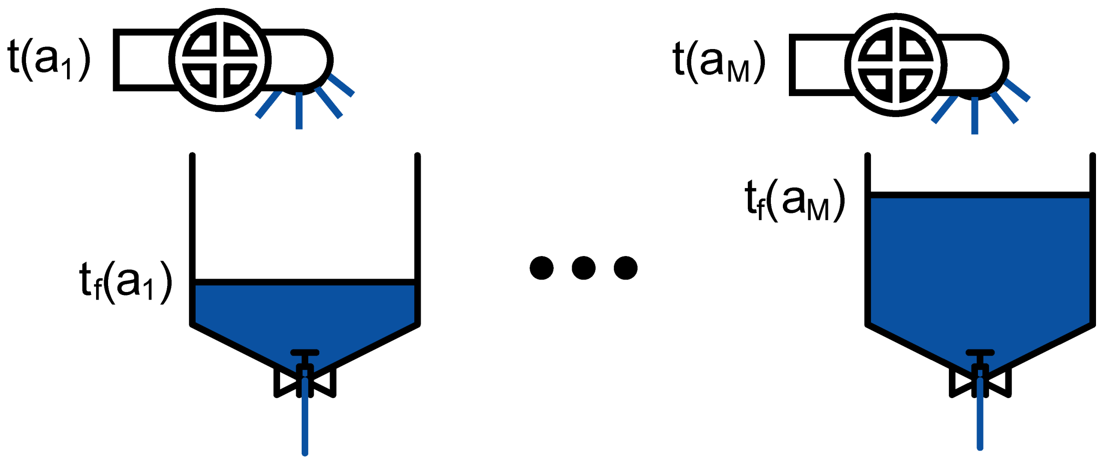


Figure 5.7: The “psycho-hydraulic” action selection filter.

ically depicted in Fig. 5.7. It was inspired by Lorenz’ “psycho-hydraulic model” [36, 37, 64] that tries to model the tendency for animal behaviors or drives as a combination of internal and external stimuli as well as inhibitory functions. In our model, for each action a_k the action tendency $t(a_k)$ is accumulated each cycle (the “inflow”), while a certain constant amount is deducted from the sum (the “outflow”). This can be interpreted as a water reservoir for each action being filled by the action tendency and drained by a constant discharge. The result is the filtered tendency $t_f(a_k)$, $k = \{1, \ldots, M\}$. If the reservoirs are not empty, the action with the highest “water level”, i.e., filtered tendency, is selected to be executed by the robot. After the finish of an action all the reservoirs are emptied.

5.8.1 Proactive Execution

As mentioned beforehand, the idea behind proactive execution is that the robot starts to execute a task even when it is uncertain about the true intention of the human. This action allows the robot to actively resolve the experienced ambiguity. To this end, we use four criteria to select the appropriate action:

1. The conditional entropy $H(I|A)$,
2. the expected success and the valence $E \times V$,
3. the safety requirements, and
4. the most likely action sequence.

They will be described in more detail subsequently.

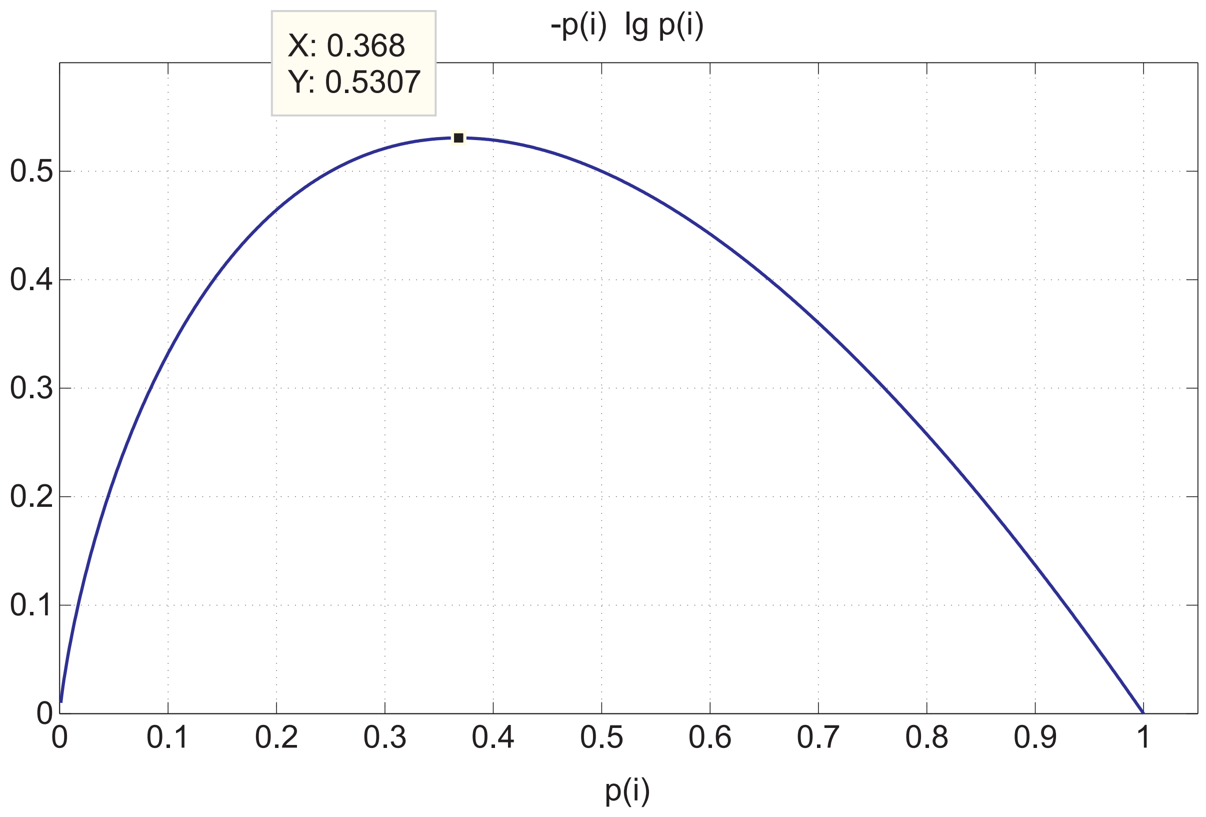


Figure 5.8: Plot of $-p(i) \lg p(i)$ over $p(i)$.

5.8.2 Conditional Entropy $H(I|A)$

The idea of this criterion is to reduce the existing uncertainty in the PDFI about the true human intention. According to information theory, we can specify the uncertainty about the human intention as the entropy of its PDF. The uncertainty u of a single intention value i_j can be defined as

$$u(i_j) = \lg \frac{1}{p(i_j)} = -\lg p(i_j) \quad , \tag{5.6}$$

where $\lg x$ is used as an abbreviation of the binary logarithm $\log_2 x$. The expected uncertainty of the whole PDFI is then

$$H(I) = -\sum_j p(i_j) \lg p(i_j) \quad , \tag{5.7}$$

which is called the entropy $H(I)$. In order to illustrate the contribution of each summation element depending on its probability $p(i)$, the value of the term $-p(i) \lg p(i)$ is plotted over $p(i)$ in Fig. 5.8. Two points are noteworthy about this graph. First, even though $\lim_{p(i)\to 0}\{-\lg p(i)\} \to \infty$, for the product $\lim_{p(i)\to 0}\{-p(i) \lg p(i)\} = 0$. The term is zero for $p(i) = 1$ as well, and positive for all other values, consequently also the entropy $H(I) \geq 0$. Second, the location of the maximum of $-p(i) \lg p(i)$ can be calculated to be at $p(i) = e^{-1} \approx 0.3679$ with a value of $(e \cdot \ln 2)^{-1} \approx 0.5307$.

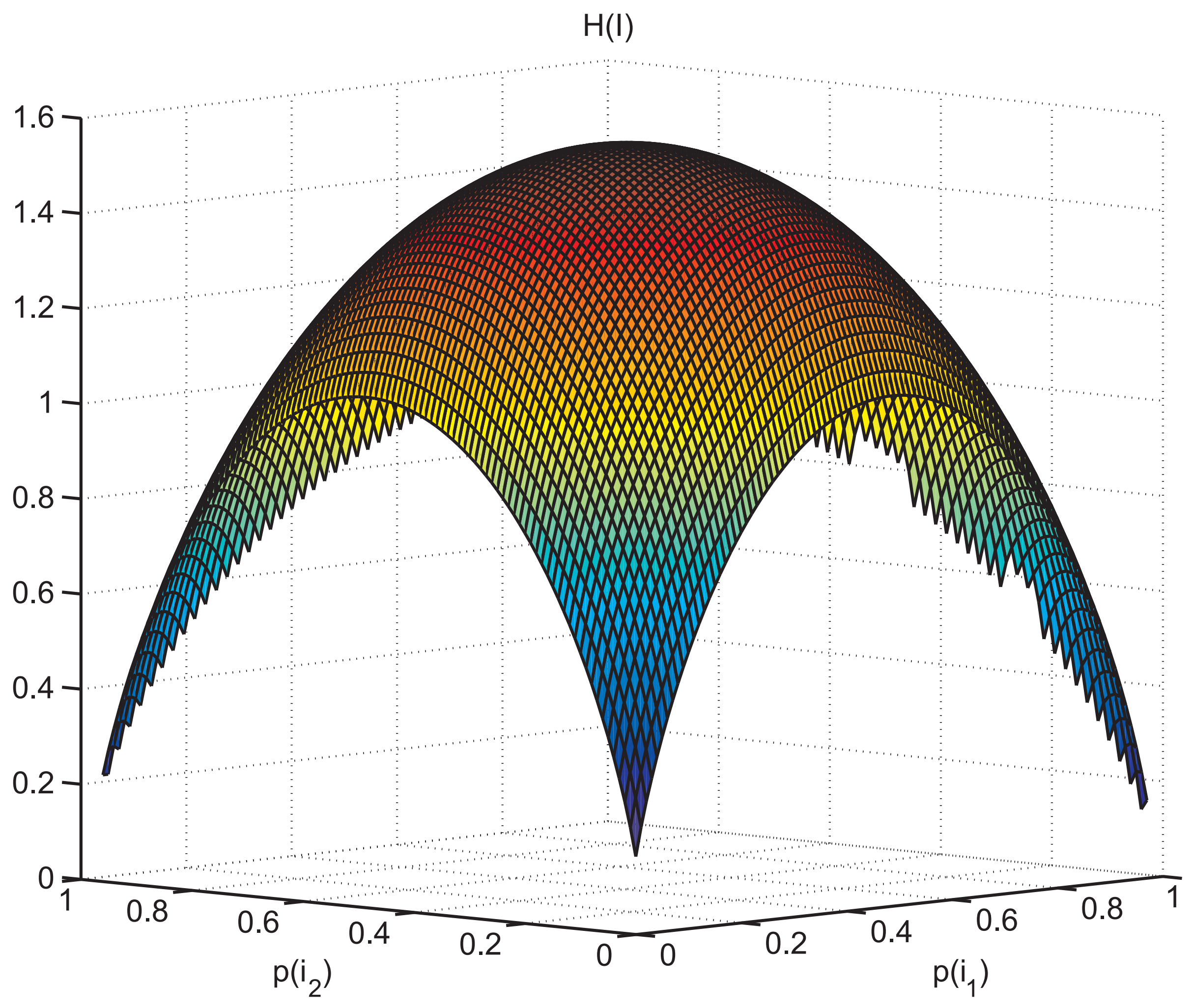


Figure 5.9: Entropy $H(I)$ plotted over $p(i_1)$ and $p(i_2)$ for $I = \{i_1,\ i_2,\ i_3\}$.

The complete entropy $H(I)$ plotted over $p(i_1)$ and $p(i_2)$ for the case of three intentions, i.e., $I = \{i_1,\ i_2,\ i_3\}$, is graphed in Fig. 5.9. It can be shown that for an arbitrary, discrete random variable $X = \{x_1,\ \ldots\ ,\ i_N\}$, the maximum entropy $H(X)$ is achieved for a uniformly distributed PDF, that is for $p(x_i) = 1/N$. The maximum is then $H(X) = \lg N$. Hence for the plot, the maximum is $\lg 3 \approx 1.5850$ at $p(x_i) = 1/3$. Putting it all together, we can state that an entropy has values of

$$0 \le H(X) \le \lg N \quad . \tag{5.8}$$

Since we are not interested in the uncertainty about the intentions $H(I)$ itself, but want to get a clue about which action will maximally reduce this uncertainty, we need to be more specific by incorporating the robot action. After selecting an action, the uncertainty of our system is reduced to the conditional entropy $H(I|A)$. We calculate $H(I|A)$ as

$$H(I|A) = -\sum_k p(a_k) \sum_j p(i_j|a_k) \lg p(i_j|a_k) \quad . \tag{5.9}$$

However, [5.9] poses a problem because the value of $p(i_j|a_k)$, that is the probability of user intention i_j given robot action a_k, is unknown. We have two ways of dealing with this. The first is using Bayes' rule to express the unknown $p(i_j|a_k)$ with the known $p(a_k|i_j)$ and thus obtain

$$\begin{aligned} H_1(I|A) &= -\sum_k p(a_k) \sum_j \frac{p(a_k|i_j)\,p(i_j)}{p(a_k)} \lg \frac{p(a_k|i_j)\,p(i_j)}{p(a_k)} \\ &= -\sum_k \sum_j p(a_k|i_j)\,p(i_j) \lg \frac{p(a_k|i_j)\,p(i_j)}{p(a_k)} \quad . \end{aligned} \qquad [5.10]$$

For $p(i_j|a_k)$ we can now simply use the values from the action selection table (Table 5.1 on p. 81) to arrive at a value $H_1(I|A)$.

The second method is to estimate the true value of $p(i_j|a_k)$. This can be done by averaging the values of the PDFI, $p(I)$, after the robot has started to execute a certain action, weighted by a factor that declines over time. The resulting $p(\bar{I})$ is then used to compute a new estimate:

$$p_{new}(\widehat{I|a_k}) = \frac{1}{\alpha+\beta}\left(\alpha\, p(\bar{I}) \;+\; \beta\, p_{old}(\widehat{I|a_k})\right) \quad . \qquad [5.11]$$

Now we can use again the original equation [5.9], which leads to

$$H_2(I|A) = -\sum_k p(a_k) \sum_j p(\widehat{i_j|a_k}) \lg p(\widehat{i_j|a_k}) \quad . \qquad [5.12]$$

For the final decision, both $H_1(I|A)$ and $H_2(I|A)$ are combined to $H(I|A)$ using

$$H(I|a_k) = \frac{H_1(I|a_k) + H_2(I|a_k)}{2 \lg N} \quad . \qquad [5.13]$$

The choice of denominator stems from [5.8] where $\lg N$ was found to be the maximum value of an entropy $H(X)$. As a consequence, $2\lg N$ normalizes the result to a value of $0 \le H(I|a_k) \le 1$.

By computing the values of H for all possible actions and comparing the results, we are able to determine the action $\breve{a}$ that has the lowest conditional entropy value and on that account leaves us with the least uncertainty, that is

$$\breve{a} = \arg_A \min H(I|A) \quad . \qquad [5.14]$$

5.8.3 Expected Success and Valence $E \times V$

The second criterion was influenced by a number of existing theories on rational decision making that were developed in the scientific discipline of ethology [62], which studies human and animal behavior, and psychology [23]. The criterion strives to find the robot action that maximizes the product of the *expected successful completion*, E, of an action and its *valence*, V. Con-

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1	1	0.5	0.5	0.5	0.5	0.5	0.5

Table 5.2: Specification of the static value $v(a_k)$.

	a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
a_1								
a_2			1	1				
a_3		1			1			
a_4		1			1			
a_5			1	1				
a_6								
a_7								
a_8								

Table 5.3: Specification of the action energization $ae(a_k, a_i)$.

sequently the method is related to a partially observable Markov decision process (POMDP), although a simplified one with a limited look-ahead horizon of one time step and the partially observable states being the set of different human intentions in the PDFI.

The expected successful completion E can be estimated by recording the execution history of each action a_k and whether it completed successfully or was aborted by the human user. The human can abort an action by saying “Stop!”, issuing the tactile `'stop'` command (cf. Fig. 4.10(f) on p. 53) or acting in such a way that the PDFI changes. We record the overall history of the actions and the history depending on the possible intentions, and hence receive 3 success probabilities e_1, e_2, and e_3. Taking the average, we get

$$E = \frac{e_1 + e_2 + e_3}{3} \qquad [5.15]$$

We chose to divide the valence V of an action a_k up into a static and a dynamic component. The static value of an action, $v(a_k)$, is defined by an expert with a table as shown in Table 5.2.

The dynamic value is represented by the *inhibitory gain*, $ig(a_k)$, of an action [7]. The greater $ig(a_k)$, the more the other actions are suppressed, and the more likely the current action stays active. Initially, ig is zero for all actions. During operation, the inhibitory gain is updated according to a pre-specified action energization table as shown in Table 5.3 that correlates the current action a_k (rows) and subsequent actions a_i (columns) with values $-1 \leq ae(a_k, a_i) \leq 1$. If there is no correlation, the value $ae(a_k, a_i)$ is zero (zeroes are omitted in Table 5.3). If the value is positive, the action a_i is energized; if it is negative, a_i is inhibited.

To make this mechanism plausible let’s look at a real-world example: Imagine the robot is currently busy with the task `'do laundry'`. According to our expert knowledge, the laundry will have to be taken out of the washing machine when it’s done. For this reason, it makes

a_1	a_2	a_3	a_4	a_5	a_6	a_7	a_8
1	1	0.5	0.5	0.85	0.5	0	0

Table 5.4: Specification of the risk value $risk(a_k)$.

sense that `'do laundry'` energizes, or activates, the task `'take laundry out of washing machine'`.

At the start of an action a_k, a new inhibitory gain value ig for each action is computed by

$$ig_{n+1}(a_i) = \frac{\alpha \cdot ae(a_k, a_i) \ + \ \beta \cdot ig_n(a_i)}{\alpha + \beta} \quad , \qquad [5.16]$$

with the parameter values currently chosen to be $\alpha = 1$ and $\beta = 2$. The total valence $V(a_k)$ of an action can then be determined by adding $v(a_k)$ and $ig(a_k)$ and limiting the value within the range of zero and one:

$$V(a_k) = \min(0, \ \min(v(a_k) \ + \ ig(a_k), \ 1) \,) \ . \qquad [5.17]$$

5.8.4 Safety Requirements

Executing a task proactively means that there is no explicit user command that selected this task, but rather that the robot is acting on an implicitly perceived order that is associated with uncertainty and thus requires the robot to make an educated guess. This automatically involves the chance that the human user expected a different task to be performed and therefore aborts the current one, or that the task generates an exception on its own and needs to interrupt itself. Depending on the nature of the task, such an interruption can be more or less of an issue. Take, for example, the pouring of a cup of hot coffee—it would be highly undesirable if the robot froze its actuators in the middle of the pouring process; rather, it should stop pouring and replace the coffee pot at a safe spot.

For this reason, this criterion takes into account the risk that an interrupted task poses to the robot's environment, especially the human. To that end, a risk value $risk(a_k)$ is assigned to each task by an expert as depicted in Table 5.4, where $0 \leq risk(a_k) \leq 1$ with greater values indicating a higher risk. By favoring a task that poses a low risk, we aim at making the robot trustworthy and safe to work with.

5.8.5 Action Sequences

The last criterion that is used to compute the action tendency in the proactive execution case aims to learn sequences of actions on-line. To this end, a FIFO memory stores the sequences of successfully completed robot actions over time (cf. Table 5.5). As such, it represents an episodic memory of the robot concerning its actions. In every cycle during a proactive exe-

Index	n	n-1	n-2	n-3	n-4	n-5	n-6	n-7	...	n-L-3	n-L-2	n-L-1
Action	7	6	8	6	8	3	5	5	...	1	5	2

Table 5.5: FIFO storage of length L at time n storing successfully completed actions.

cution, this memory is searched for action sequences in the past that are equal to the present action sequence, i.e., the actions that were recorded last in the memory. If a match is found, the action a_k that followed this sequence in the past can be determined. Then, the variable $seq(a_k)$ is incremented by the length of the matched sequence multiplied by a weighting factor w that decreases for actions that are further in the past:

$$seq = \sum_{i \in matches} w_i * length(sequence_i) \ . \qquad [5.18]$$

The resulting value of $seq(a_k)$ indicates how likely action a_k will follow in sequence.

5.8.6 Tendency Accumulation

Putting it all together, we arrive at the following equation to calculate the action tendency for an action a_k:

$$t(a_k) = \frac{1}{2}\phi(1 - H(I|a_k)) \ + \ \frac{1}{4}E(a_k)\,V(a_k) \ + \ \frac{1}{2}(1-\phi)(1 - risk(a_k)) \ + \ \frac{1}{4}seq(a_k) \qquad [5.19]$$

The factor $\phi \in [0.1, 0.9]$ serves as a control to render the action tendency more conservative (low values of ϕ) or more exploratory (high values of ϕ). This is true because the first criterion, conditional entropy, tends to favor uncommon actions with a lower probability since they reduce the entropy more than standard ones; consequently it can be considered an exploratory criterion. The third criterion, however, tends to favor less risky operations and therefore behaves more conservatively. At the beginning of operation, a low value $\phi = 0.1$ is chosen. It is changed in accordance with the frequency of successful task completions versus task abortions. The more tasks are completed successfully, the more exploratory a tendency is chosen. Each of the four criteria for action selection can contribute a factor in $[0, 1]$. Ergo if one chooses $\phi = 0.5$ the criteria are weighted equally. A graphical view of all the components forming the action tendency $t(a_k)$ is given in Fig. 5.10.

5.9 Experimental Results

5.9.1 Software and Hardware

As already mentioned in section 3.2 we use the freely available Modular Controller Architecture (MCA) as a software framework for the entire robot control architecture. In order to test and

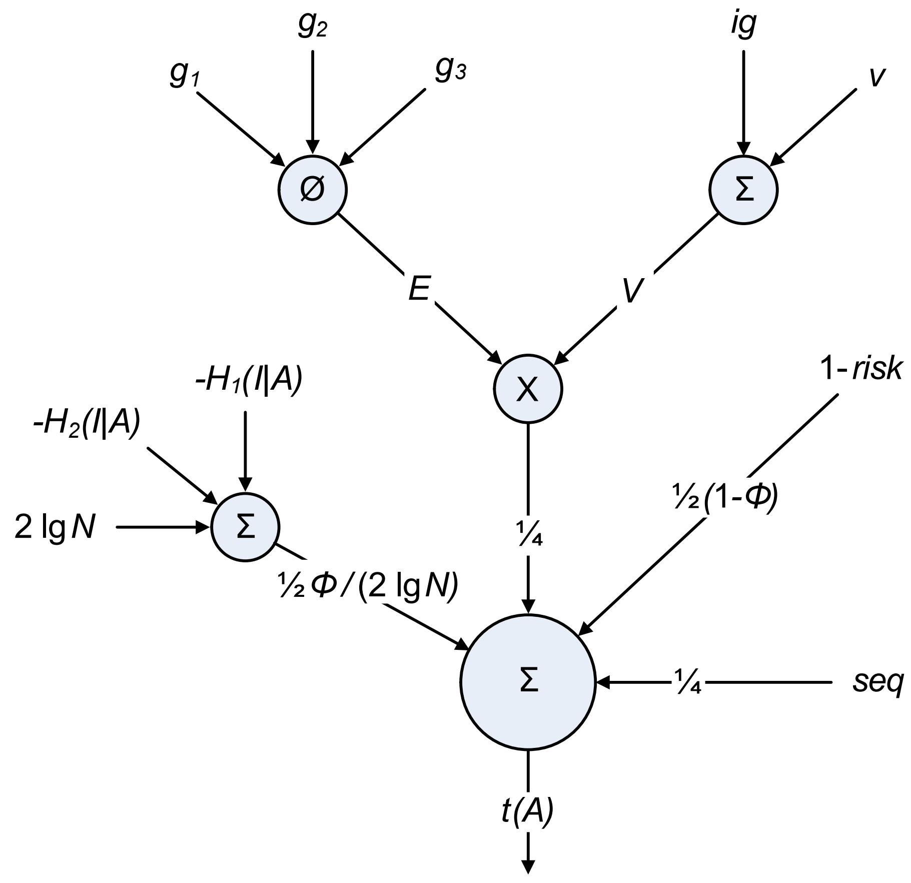


Figure 5.10: The computation of the action tendency $t(a_k)$ in the proactive execution case.

demonstrate the proactive execution, MCA2's graphical user interface (GUI) tool was adapted: The GUI can be freely fashioned according to one's specific needs by selecting the desired widget types and pulling them into place on the surface. Figure 5.11 presents the GUI that we assembled for the experiments. On the top left it shows the momentary action tendencies, on the top middle the PDFI. On the right the probabilities of the cup and the plate being grasped are displayed. The bottom left button indicates the proactive execution case, i.e., the number of possible intentions is between one and `PE_MAX`. Right next to it the context is indicated as explained below. Finally, the person and object tracker windows are both displayed; the two trackers work with the same picture stream from the stereo camera on top of the robot.

As to the robot hardware, you can see a picture of our current evaluation platform in Fig. 3.1 on page 28. Eventually, our modules will run on the demonstrator robot of our collaborative research center shown in Fig. 1.2 on p. 2.

5.9.2 Intention Recognition and Proactive Execution demonstration modules

The current version of the intention recognition software has been integrated into the robot control as a C++ library. The library functions are called from within an MCA module that

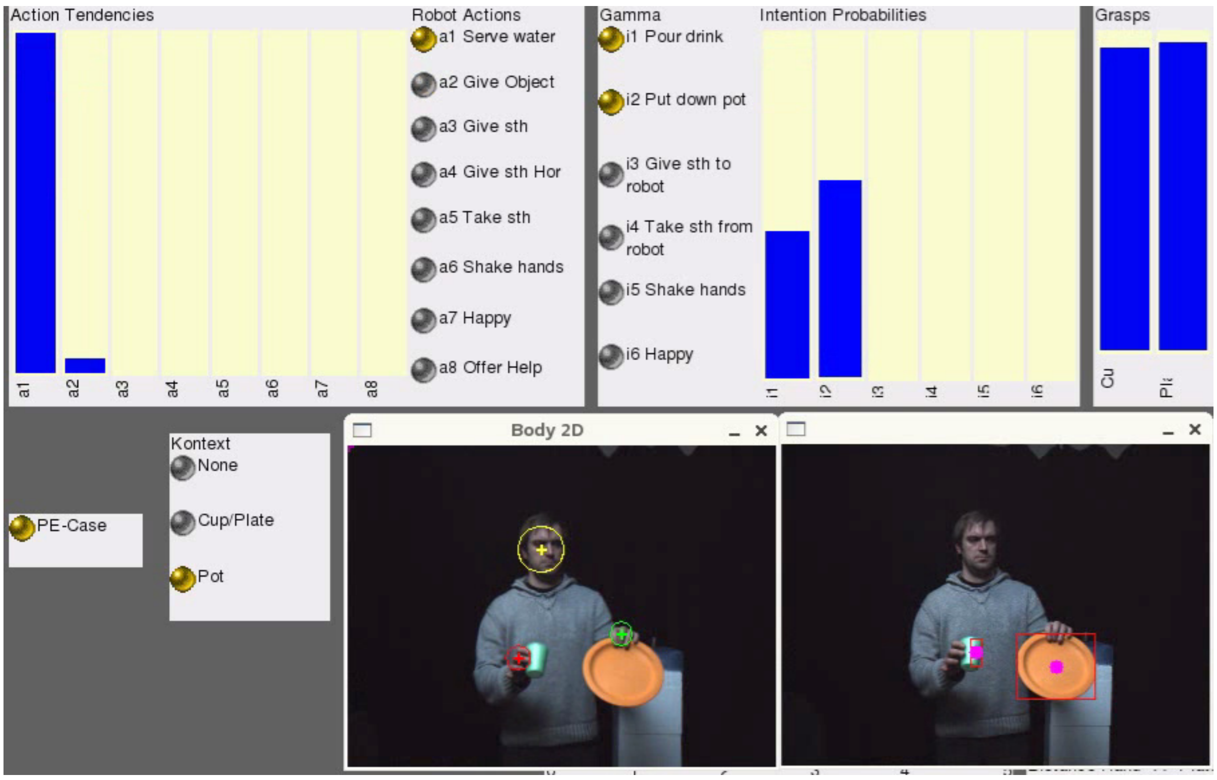


Figure 5.11: The GUI that was used for the experiments.

is connected with all available perception modules for input values, and the proactive execution module that receives the output as a probability density vector over the human intentions. Though designed for maximum flexibility as far as the perception of the environment is concerned, the intention recognition currently relies solely on vision through the robot's stereo camera system as sensor input. Specifically, it uses two modules that were developed in our collaborative research center—Kai Nickel's person tracker [44] that tracks one person's head and two hands, and Pedram Azad's object tracker [3] with which we recognize pots, cups and plates based on color segmentation. Whether an object is grasped by the human or not is determined by the distance of the human's hands to that object; the actual probability value of the object's being grasped is computed using a probability density function generated by a Gaussian mixture density that has a maximum value up to a distance of 20 cm and then gradually reaches zero for larger distances. Finally, the Bayes net intention estimator is influenced by the situation context. In the demonstration scenario case, the context refers to what the robot is holding in its hand. There are three possibilities that are differentiated:

- nothing,
- a cup or a plate, or
- a pot.

I	*Description*	*A*	*Description*
i_1	`Pour drink`	a_1	`Serve water`
i_2	`Put down pot`	a_2	`Place pot`
i_3	`Give object to robot`	a_3	`Take object from user vert.`
		a_4	`Take object from user hor.`
i_4	`Take object from robot`	a_5	`Give object to user`
i_5	`Shake hands`	a_6	`Shake hands`
i_6	`Happy`	a_7	`Happy`
		a_8	`Offer help`

Table 5.6: Intentions and actions of the evaluation scenario.

Based on this input, the intention recognition estimates the human intention considering six alternatives. They are listed in Table 5.6 in the left column.

The proactive execution module was completely implemented and works as described above in sections 5.4 to 5.8. The actions to select from are listed in Table 5.6 in the right column, and their mapping to intentions additionally in Table 5.1. Hence there are 8 actions for 6 intentions. The result is a suggestion to the central planner of the robot control architecture as to what task to execute next. Our system is realtime capable. We use a loop interval of 100 ms for computing the PDFI, i.e., the $p(I)$, the action tendencies $t(A)$, and for deriving an action.

For evaluation purposes, we ran the demonstration scenario and recorded the relevant data. There is no predetermined action sequence the human experimenter has to adhere to as far as the actual human-robot interaction is concerned. The basic conditions under which the individual intentions are recognized are as follows. If the robot's hand is empty and the human-robot distance is about 1.8 m or more, it is assumed that the human is `'happy'` and does not wish to interact with the robot. As the human comes closer to the robot and is empty-handed, the probability of the intention `'shake hands'` increases. If he carries an object, though, the intention `'give object to robot'` is deemed likely. After such a handover of an object between human and robot, the context changes. If the robot is holding a cup or a plate and is approached by a human, the robot assumes that the human would like to take over the object from the robot, i.e., `'take object from robot'`. When the context is such that the robot holds a pot, two other intentions come into play. In case the human approaches the robot with a cup, then intention `'pour drink'` should be recognized; in case the human holds a plate, on the other hand, the robot figures that the human wants it to place the pot on the plate (`'put down pot'`).

5.9.3 First Sample Section

Figure 5.12 shows the progress of the intention probabilities during the first 66 seconds of the experiment. As the graph title indicates, the first graph displays the raw values as computed by

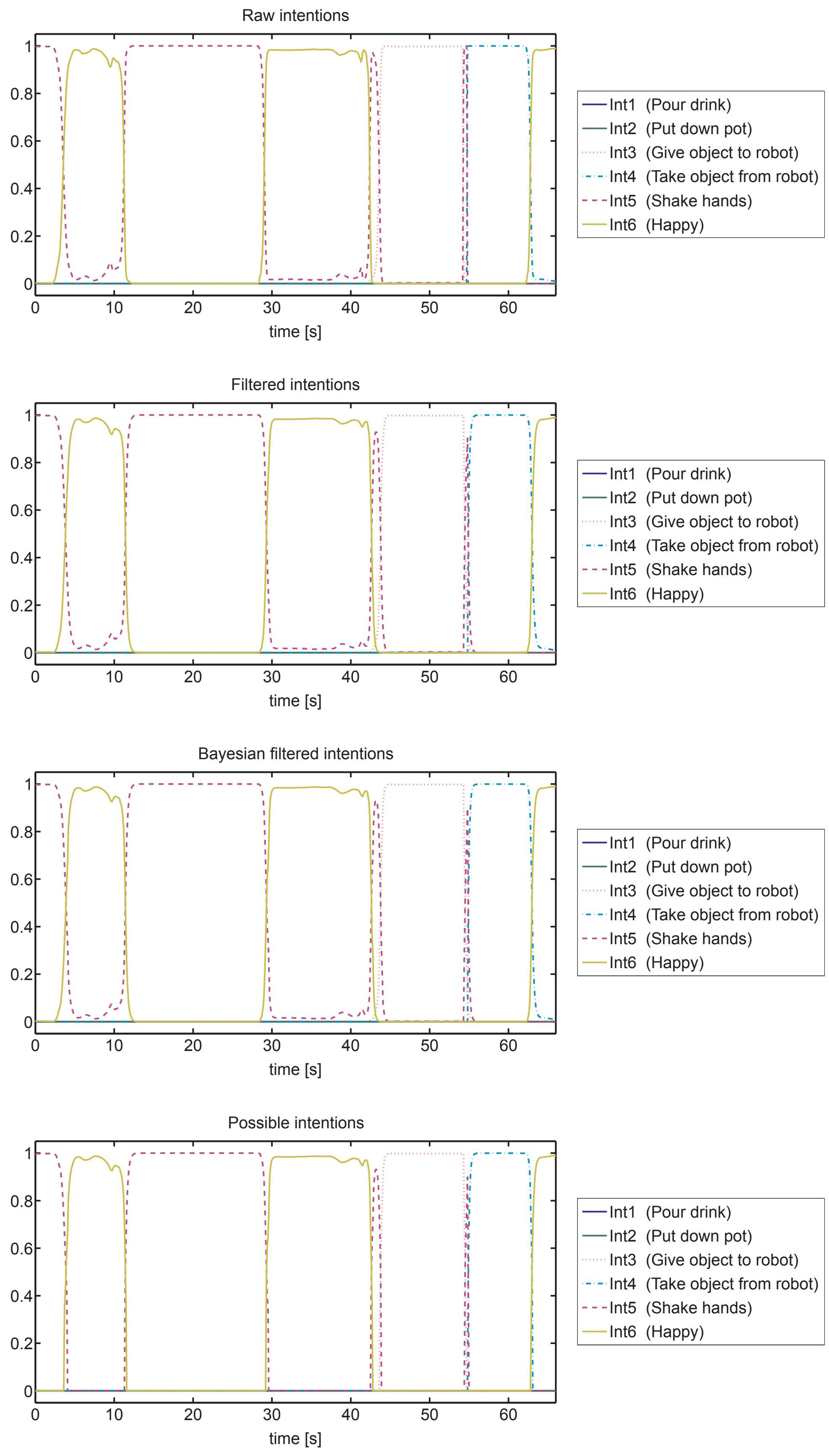


Figure 5.12: Progress of the intention probabilities—raw data and after each processing step.

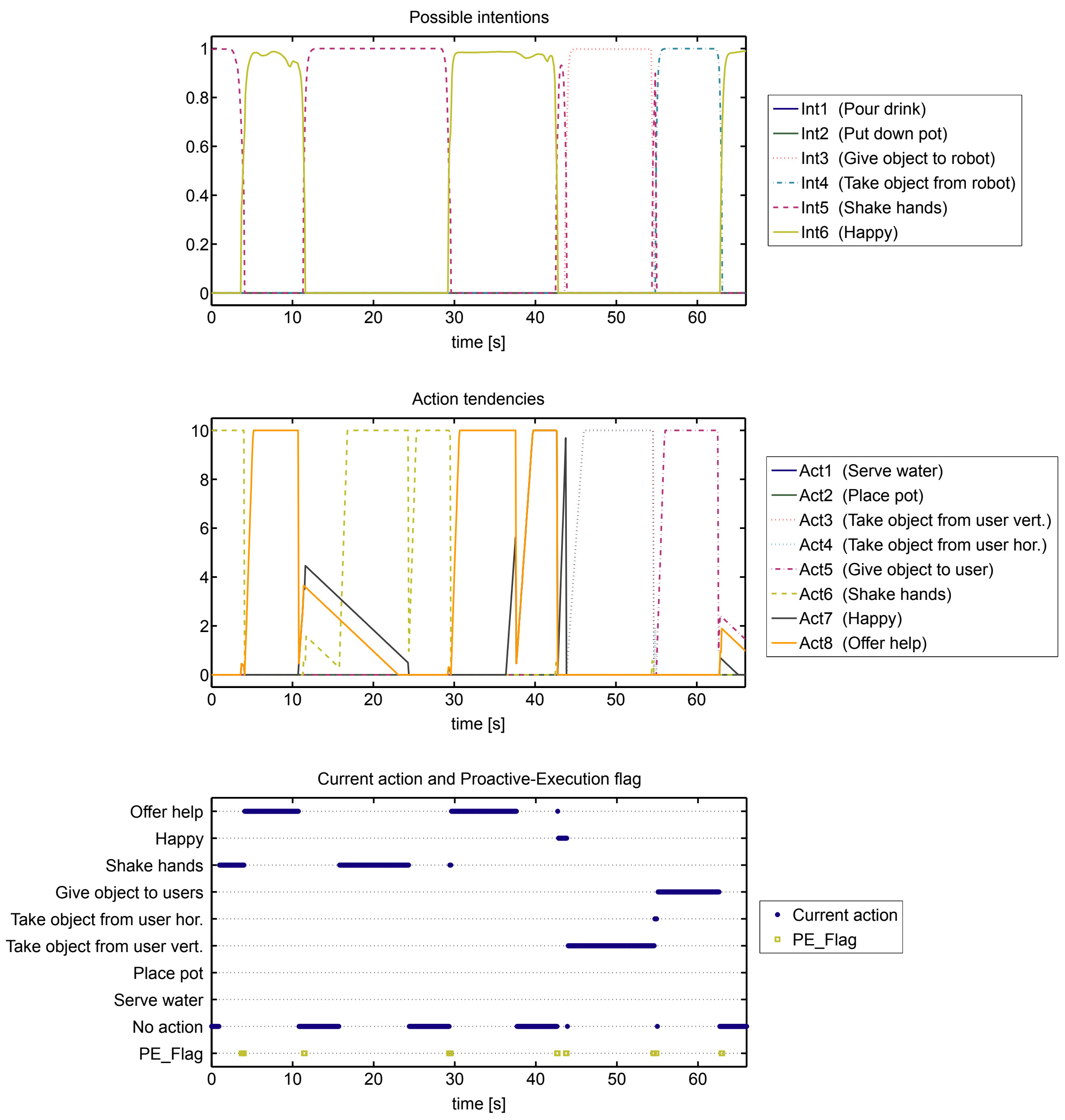


Figure 5.13: Progress of the action tendencies and currently executed actions.

the intention recognition. You can see that the intentions are recognized pretty clearly without much ambiguity. The basic sequence of dominant intentions is `'happy'` → `'shake hands'` → `'happy'` → `'give object to robot'` → `'take object from robot'` → `'happy'`. Due to the unambiguity of the experimental conditions during this phase of the experiment, and also due to the large scale of 66 seconds that correspond to 660 measurement points, the results of the two filtering steps that are shown in the second and third graph are not very different from the raw data. The main effect is the reduction in amplitude of the `'shake hands'` spikes at $t = 42.8$ s and $t = 54.5$ s. The last graph at the bottom shows the current possible intentions with their current values while the other intentions are set to zero.

Figure 5.13 shows the sequence of the possible intentions (top graph) and the corresponding action tendencies (middle graph) that were determined by the proactive execution module. The bottom graph shows the currently selected action and also whether the execution was labelled as proactively (`PE_Flag`). The middle graph illustrates how a change of the possible intention values causes a subsequent ramp-up or decline of corresponding action tendencies. The gradual ramp-up is due to the limited amount of action tendency that is added to the accumulated tendency during each cycle. Another noteworthy point are the sudden action tendency drops at $t = 10.7$ s, $t = 24.3$ s, $t = 37.6$ s, and $t = 62.6$ s. At those times the execution supervisor reported that the current action had successfully finished. As a consequence of such a report, all accumulated action tendencies are reset to zero. Moreover, this particular action is blocked for `FINISHED_BLOCK_TIME` (currently set to 5 s); this blockage was introduced to give the human a chance to indicate a different intention after a successful task completion and consequently avoid repeating the same action unintendedly. Should the user actually wish to repeat the action, on the other hand, he can keep his current behavior which will re-trigger the action in due course. This system design point explains why the action tendencies do increase again after a reset with persistently high values of the same intention but the selected action remains at `'no action'`.

5.9.4 Second Sample Section

The just discussed phase of the experiment served well to explain the basic behavior of the whole system but did not contain any proactive behavior. Hence another section of the experiment is examined as shown in Fig. 5.14. It covers the time $129\,\mathrm{s} \leq t \leq 141$ s, providing a higher resolution than Fig. 5.13 such that individual data points are visible. The situation that is reflected by the data has the robot holding a pot and the human in front of the robot with a cup in one hand and a plate in the other. Depending on the distances between the robot and the grasped objects, `'pour drink'` and `'put down pot'` are recognized as human intention with corresponding probability. Looking at the second graph, it becomes evident that in this section the filter step successfully smoothes the course of the intentions such as at positions $t = 129.8$ s and $t = 139.3$ s. Apart from a little smoothing effect at $t = 129.5$ s for intention i_4 (`'take object from robot'`), the Bayesian filter step doesn't change much in the third graph. We credit this to the lack of history and to similar average values for the intentions in question. The bottom graph showing the possible intentions indicates several regions of overlapping intentions, i.e., sections where more than one intention was found to be possible.

Looking at the associated action tendencies and the executed actions in Fig. 5.15 reveals the potential of the proactive execution strategy. Whenever there is an ambiguous situation marked by the golden `PE_Flag` on the bottom line in the bottom graph, the current action tendencies

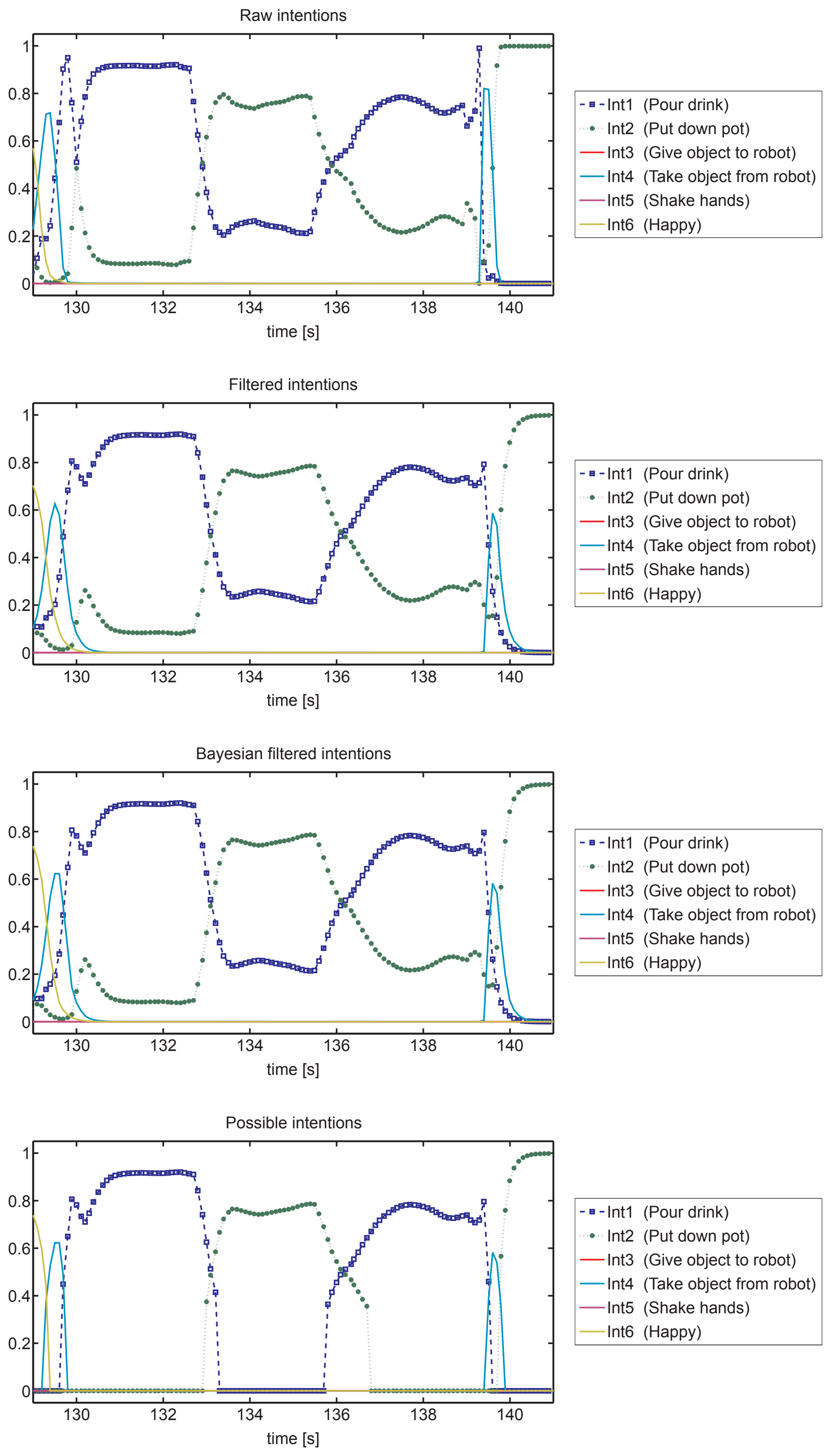


Figure 5.14: Progress of the intention probabilities at the second section.

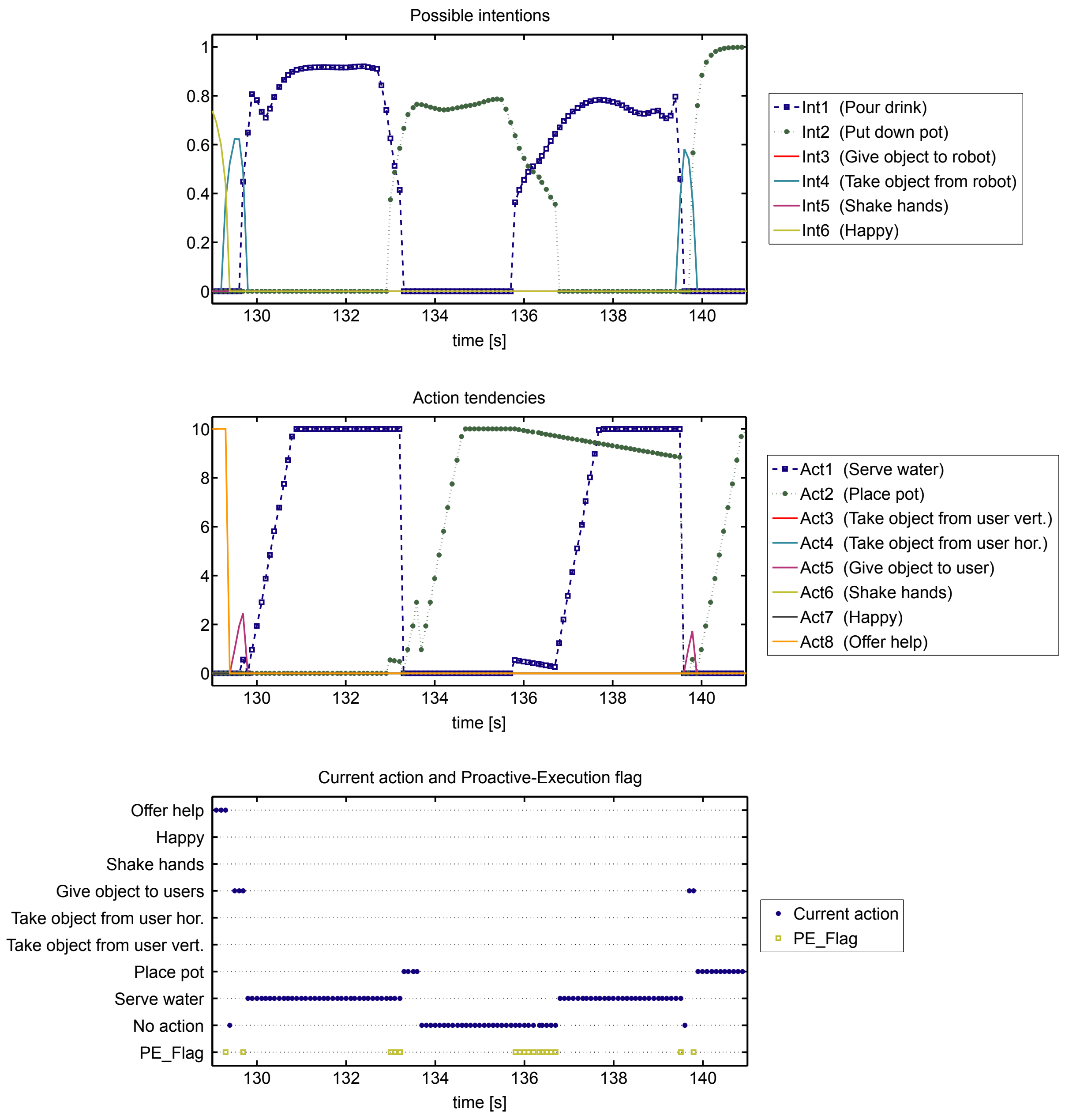


Figure 5.15: Action tendencies and currently executed actions at the second section.

provide a good basis for the actual selection. The figure also reveals a bug in the control logic in that the `place pot` action that should be executed according to its tendency between $t = 133.3$ s and $t = 137.6$ s is in fact blocked and `no action` selected. A more detailed analysis shows that the `finished` signal for the previous action, `serve water`, was interpreted as belonging to the current action `place pot`, leading to its termination and being blocked.

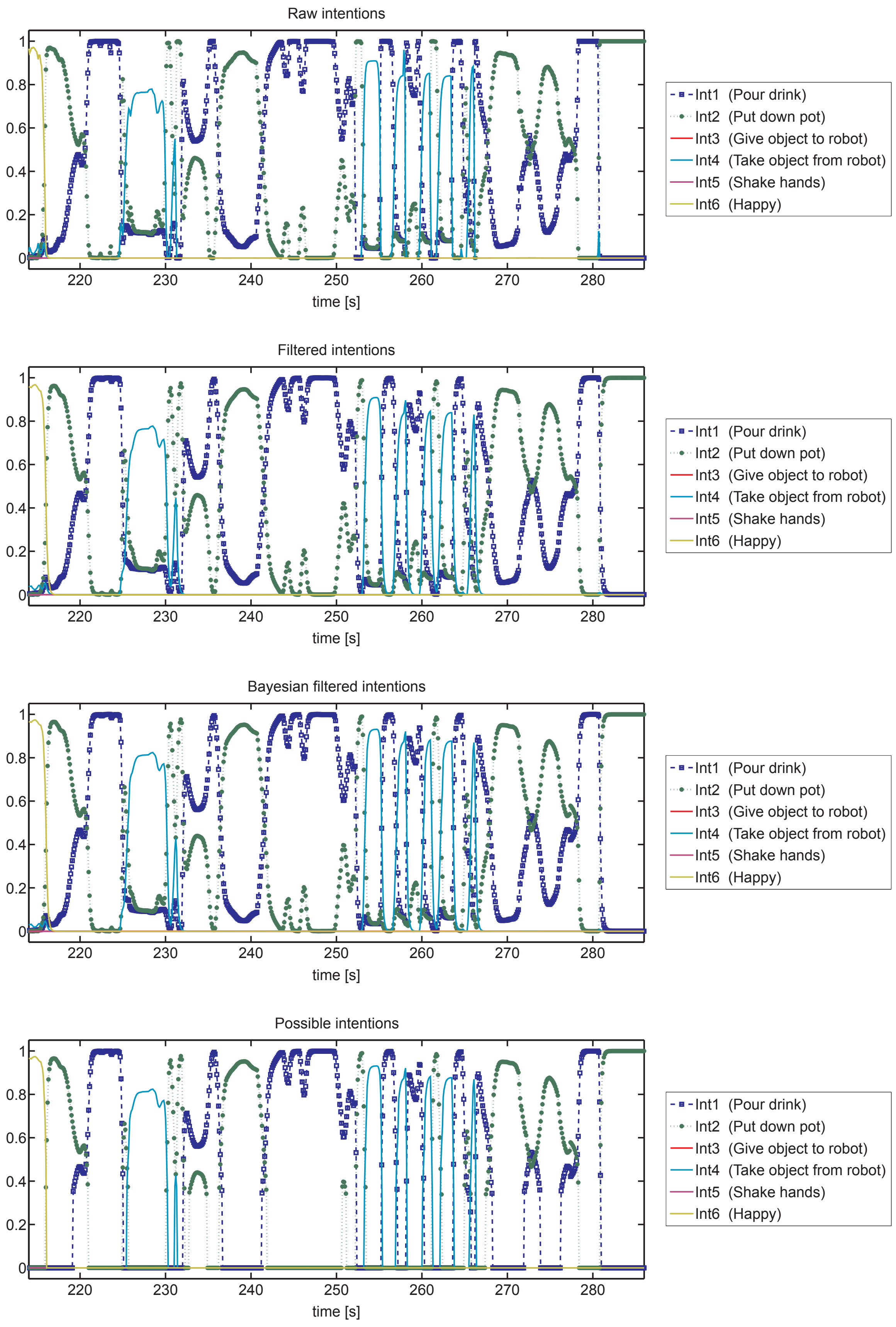


Figure 5.16: Progress of the intention probabilities at the third section.

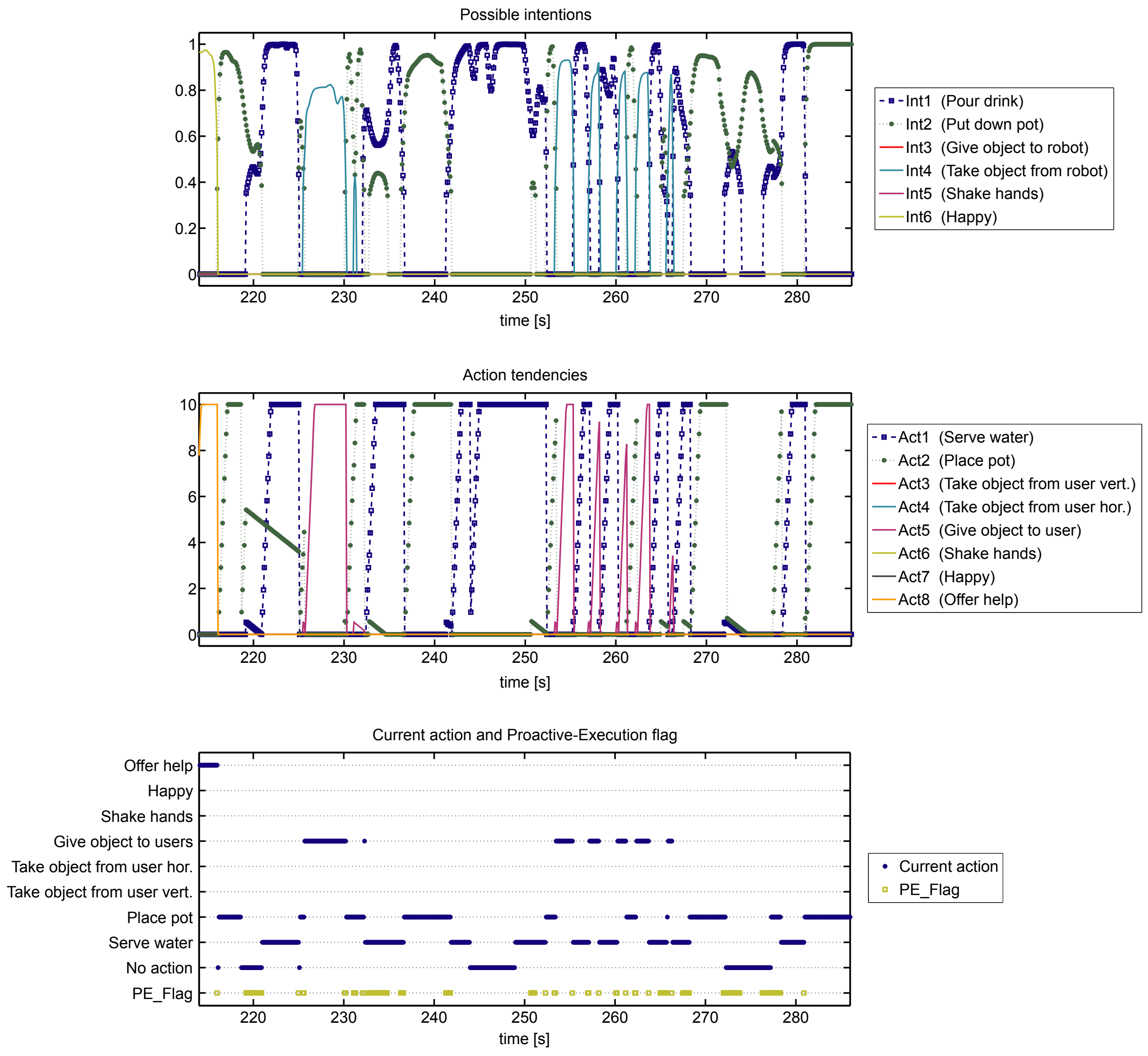


Figure 5.17: Action tendencies and currently executed actions at the third section.

5.9.5 Third Sample Section

Figures 5.16 and 5.17 show a third and last sample section at $214\,\mathrm{s} \leq t \leq 286\,\mathrm{s}$. They exhibit a more rapid succession of changes in the PDFI, and how the action tendencies follow. The action that was selected for execution at any given time is presented again in the bottom graph of Fig.5.17. Action tendencies and selected actions match very well, and together they resolve any ambiguities that exist in the possible intentions (top graph). Obviously, though, the actually selected action changes quickly, too, despite the smoothing effects of the proactive execution algorithm. If the robot would follow every change, it would act rather erratically. This is highly undesirable, as we want the user to trust and be comfortable with the robot. For this reason, we adjusted the design of the execution supervisor such that tasks with finite duration are not pre-empted by newly suggested tasks from the proactive execution module. This holds true unless the `PE_Flag` is active, indicating that the current action was started proactively. In that case, we might have selected the wrong task after all, and thus we want to be able to correct that based on what we learn from the user reactions and ensuing changes in the PDFI. According to our experiences it is virtually impossible to find a design solution that satisfies all needs in every situation. The more autonomous the robot becomes, the less influence the human possesses over the robot, and the more likely the robot is to do the wrong thing. On the other hand, less autonomy requires more human-robot interaction which can become tedious very quickly for repetitive tasks.

5.10 Summary

This chapter introduced the concept of the proactive execution module that forms an integral part of our intuitive human-robot cooperation system. It detailed its action selection process, and listed the four specific selection criteria for the case of the proactive execution of a robot action. Lastly, details of the implementation and the demonstration scenario were given and their results examined.

6 Conclusions and Outlook

In this chapter we summarize the contributions that this thesis makes, and give an outlook on future improvements that could be done in continuation of this research.

6.1 Contributions

Robot Architecture This thesis presented a robot architecture for a service robot in a human-populated environment. The architecture was designed to enable a natural and intuitive human-robot interaction with an untrained human by facilitating the integration of arbitrary man-machine interfaces (MMIs) such as natural and tactile language, and a proactive execution module for implicit commanding of the robot.

An essential requirement for the safe and acceptable operation in a human environment such as a kitchen is that the robot acts dependably, reliably and predictably, and that it leaves the human in control. The architecture achieves this by giving the user's explicit commands the highest priority and making the proactively executed tasks interruptible on user request. Thus the robot's autonomy is restricted to a comfortable measure.

In addition, a mechanism was incorporated to detect tasks that fail to make progress and thus block the robot system. These tasks are aborted to clear the way for new instructions to execute.

This architecture was implemented on our test bed robot and used successfully to test and demonstrate the tactile language and the proactive execution interfaces evaluate its performance.

Tactile Language With the tactile language, a new MMI was conceived for controlling a robot through its artificial skin. It constitutes an interface that on the one hand can be used redundantly next to a dialog manager or visual command interpreters. On the other hand, the tactile modality provides unique properties to render human-robot interaction not only more intuitive, but also more efficient: A lot parameters like direction, distance, and speed can be decoded from a single human finger stroke and used as attributes to further define a user command.

The software architecture was designed for modularity. Thus, it allows the use of arbitrary sensors that offer a spatial resolution of the tactile input. So far, the tactile language has been used with our robot's artificial skin and a multi-touch finger pad. For maximum flexibility, the tactile language offers the user four different input modes to choose from: The first two are a direct and an indirect control mode to move and rotate the robot's TCP and head. The third is an abstract mode where symbolic, complex robot commands can be issued by entering appropriate multi-finger symbols. Furthermore, by integration of a freely available character recognition software, the user can enter handwritten alphabet characters. Lastly, there is a button mode that works like a touchscreen with active regions on the tactile surface that directly issue robot commands.

The symbol recognition only requires low computational demands, thus the system is able to work with approximately real-time performance and can be used for quick on-line control of the robot. If the user wants to extend the tactile command set, this is easily possible by editing a simple text file that describes the deterministic finite automaton (DFA) that interprets the tactile input.

The user evaluation of this tactile language system yielded encouraging results. It was proven that the symbol and handwritten character recognition work sufficiently well even for untrained users. Moreover, complex tasks involving a combination of control modes were successfully mastered by all test participants, and the repetitive execution of the missions showed that the performance was significantly increased due to a learning effect from trial to trial. Last but not least the users reported a high satisfaction with the system.

Proactive Execution A novel implicit robot command interface was introduced by the proactive execution module. It allows an intuitive human-robot cooperation in that the robot capitalizes on its estimation of the human intention based on non-verbal cues and selects an appropriate robot action. As a consequence, there is no need for explicit user commands anymore.

While the action selection is straightforward for unambiguous results of the intention recognition, it is more involved in case of high uncertainty. Instead of idling, the robot proceeds to execute tasks “proactively”. For this case, four criteria have been established to ensure the selection of a sensible and purposeful action: The first one seeks to reduce the uncertainty of the system as measured by the conditional entropy $H(I|A)$. The idea of the second is to maximize the expected utility $E \times V$ of the selected action. The third criterion aims at ensuring human safety by preferring actions that pose a lower risk. Finally, the fourth criterion tries to benefit from the action execution history recorded in an episodic memory by searching from action sequences in the past that resemble the current one. Overall, through this proactive execution mode, the robot is enhanced from a machine that merely takes orders to an autonomously acting assistant.

The proactive execution system has been implemented and was adapted to a demonstration scenario. In this process it proved to scale well to any reasonable number of recognized human intentions and available robot actions, and changes in these environment parameters are compensated quickly by a few table updates. The functionality of the proactive execution module was successfully demonstrated and encourages further work in this area.

6.2 Outlook

Both presented human-robot interfaces and the corresponding robot architecture were implemented on our humanoid robot manipulator shown in Fig. 3.1 on p. 28. Obviously, the absence of a mobile base and a second arm constitute severe limitations on the interfaces' and the robot's usability, at least in the long run. In the future, it is absolutely desirable to transfer the results of this thesis onto a full-featured humanoid robot such as *ARMAR III* (Fig. 1.2 on p. 2). In this process, the usefulness of the tactile language could be evaluated when entered directly on the robot's artificial skin, provided that the sensor has been equipped by an appropriate low-friction surface.

As far as the proactive execution is concerned, it would be beneficial to enlarge its basis for action selection. Currently, only the human intention estimate is used. This should be expanded to include the whole context that the robot is aware of—the current situation, knowledge about the environment, and also its own robot status. Given this information, a stochastic, model-predictive decision process could be employed to further improve the quality of the robot's action selection, and thus its usefulness to the human user.

A List of Figures

B List of Tables

C Bibliography

[1] Apple Inc. iPhone User's Guide. *http://www.apple.com/iphone/*, accessed 19/03/2008.

[2] G. Armano, G. Cherchi, and E. Vargiu. An Agent Architecture for Planning in a Dynamic Environment. In *Proceedings of the 7th Congress of the Italian Association for Artificial Intelligence on Advances in Artificial Intelligence (AI*IA 2001)*, 2001.

[3] P. Azad, T. Asfour, and R. Dillmann. Combining Appearance-based and Model-based Methods for Real-Time Object Recognition and 6D Localization. In *Proceedings of the International Conference on Intelligent Robots and Systems (IROS), Beijing, China*, 2006.

[4] C. Bauckhage, G. Fink, J. Fritsch, F. Kummmert, F. Lomker, G. Sagerer, and S. Wachsmuth. An Integrated System for Cooperative Man-Machine Interaction. In *Proceedings of the 2001 IEEE International Symposium on Computational Intelligence in Robotics and Automation*, Banff, Canada, 2001.

[5] R. Becher, P. Steinhaus, and R. Dillmann. The Collaborative Research Center 588: Humanoid Robots—Learning and Cooperating Multimodal Robots. In *Proceedings of Humanoids 2003, Karlsruhe, Germany*, 2003.

[6] R. Becher, P. Steinhaus, and R. Dillmann. The Collaborative Research Center 588: Humanoid Robots—Learning and Cooperating Multimodal Robots. *International Journal of Humanoid Robotics*, 1(3):429–448, 2004.

[7] B. Blumberg. Action-Selection in Hamsterdam: Lessons from Ethology. In *Proceedings of the Third International Conference on Simulation of Adaptive Behaviour*, pages 108–117. MIT Press, Cambridge MA, 1994.

[8] J. Bouchet, L. Nigay, and T. Ganille. The ICARE component-based approach for multimodal input interaction: Application to real-time military aircraft cockpits. In *Conference on Human Factors in Computing Systems*, 2004.

[9] C. Breazeal. Robots in Society: Friend or Appliance? In *Agents99 Workshop on Emotion-based Agent Architecture*, pages 18–26, Seattle, WA, 1999.

[10] R. A. Brooks. A Robust Layered Control System for a Mobile Robot. A. I. Memo 864, MIT, 1985.

[11] C. Burghart, R. Mikut, H. Holzapfel, T. Asfour, R. Stiefelhagen, P. Steinhaus, and R. Dillmann. A Cognitive Architecture for a Humanoid Robot: A First Approach. In *Proceedings of Humanoids 2005, Tsukuba, Japan*, 2005.

[12] N. Chomsky. Three models for the description of language. *IEEE Transactions on Information Theory*, 2(3):113–124, Sep 1956.

[13] P. R. Cohen, M. Johnston, D. McGee, S. Oviatt, J. Pittman, I. Smith, L. Chen, and J. Clow. Quickset: Multimodal interaction for distributed applications. In *Proceedings of the Fifth International Multimedia Conference (Multimedia '97)*, 1997.

[14] E. Coste-Manière and R. Simmons. Architecture, the Backbone of Robotic Systems. In *Proceedings of the IEEE International Conference on Robotics and Automation*, 2000.

[15] G. Dalgarno. *Didascalocophus, or, The deaf and dumb mans tutor*. Halton, Oxford, 1680.

[16] de Croon G., N. S., and P. E.O. *Complex Engineering Systems*, chapter Towards proactive embodied agents: On the importance of neural mechanisms suitable to process information in time. Perseus Books Groups Press, 2004.

[17] E. Gat. On Three-Layer Architectures. In D. Kortenkamp, R. P. Bonnasso, and R. Murphy, editors, *Artificial Intelligence and Mobile Robots*. MIT/AAAI Press, 1997.

[18] N. Gorges, A. Schmid, D. Osswald, and H. Wörn. A Framework for Creating, Coordinating, and Executing Skills on a Humanoid Robot. In *Proceedings of the 2007 IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, 2007.

[19] G. Grunwald, G. Schreiber, A. Albu-Schaffer, and G. Hirzinger. Programming by Touch: The Different Way of Human-Robot Interaction. *Industrial Electronics, IEEE Transactions on*, 50(4):659–666, 2003.

[20] J. Gunderson. Adaptive Goal Prioritization by Agents in Dynamic Environments. In *Proceedings of the 2000 IEEE International Conference on Systems, Man, and Cybernetics*, 2000.

[21] A. Haasch, S. Hohenner, S. Hüwel, M. Kleinehagenbrock, S. Lang, I. Toptsis, G. A. Fink, J. Fritsch, B. Wrede, and G. Sagerer. BIRON – The Bielefeld Robot Companion. In *Proceedings of the Int. Workshop on Advances in Service Robotics*, Stuttgart, Germany, 2004.

[22] S. G. Hart and L. E. Staveland. *Human Mental Workload*, chapter Development of NASA-TLX (Task Load Index): Results of Empirical and Theoretical Research, pages 239–250. Elsevier, 1988.

[23] J. Heckhausen and H. Heckhausen. *Motivation and Action*. Cambridge University Press, New York, NY, 2008.

[24] S. Hellkvist. Hellkvist.org - Software. *http://hellkvist.org/software/*, accessed 19/03/2008.

[25] S. Hellkvist. On-line character recognition on small hand-held terminals using elastic matching. Master's thesis, Royal Institute of Technology, Stockholm, Sweden, 1999.

[26] M. Hoffmann. Entwicklung einer taktilen Sprache zur intuitiven Mensch-Roboter-Kommunikation. Diplomarbeit, Universität Karlsruhe (TH), Germany, 2007.

[27] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison-Wesley, 2^{nd} edition, 2001.

[28] M.-K. Hu. Visual Pattern Recognition by Moment Invariants. *IEEE Transactions on Information Theory*, 8(2):179–187, 1962.

[29] H. Iwata and S. Sugano. Human–Robot-Contact-State Identification Based on Tactile Recognition. *IEEE Transactions on Industrial Electronics*, 52(6):1468–1477, December 2005.

[30] G. A. Kaminka, Y. Elmaliach, I. Frenkel, R. Glick, M. Kalech, and T. Shpigelman. Towards a Comprehensive Framework for Teamwork in Behavior-Based Robots. In *Proceedings of the Eigth Conference on Intelligent Autonomous Systems (IAS-8)*, 2004.

[31] A. K. Karlson and B. B. Bederson. Direct Versus Indirect Input Methods for One-Handed Touchscreen Mobile Computing. Technical Report HCIL-2007-04, Human-Computer Interaction lab, Computer Science Department, University of Maryland, MD, 2007.

[32] A. K. Karlson and B. B. Bederson. ThumbSpace: Generalized One Handed Input for Touchscreen-Based Mobile Devices. In *Proceedings of INTERACT '07*, 2007.

[33] O. Kerpa, D. Osswald, S. Yigit, C. Burghart, and H. Wörn. Arm-hand-control by tactile sensing for human robot co-operation. In *Proceedings of Humanoids 2003, Karlsruhe, Germany*, 2003.

[34] G. Kim, W. Chung, M. Kim, and C. Lee. Tripodal Schematic Design of the Control Architecture for the Service Robot PSR. In *Proceedings of the 2003 IEEE Conference on Robotics and Automation*, Taipei, Taiwan, 2003.

[35] R. Kurazume, H. Omasa, S. Uchida, R. Taniguchi, and T. Hasegawa. Embodied Proactive Human Interface "PICO-2". In *Proceedings of the 18th International Conference on Pattern Recognition (ICPR'06)*, volume 2, pages 1233–1237, Los Alamitos, CA, USA, 2006. IEEE Computer Society.

[36] K. Lorenz. The comparative method in studying innate behavior patterns. *Symposia of the Society for Experimental Biology*, 4:221–268, 1950.

[37] K. Lorenz. *The Foundations of Ethology*. Springer, 1981.

[38] K. Madhava Krishna, R. Alami, and T. Simeon. Safe proactive plans and their execution. *Robotics and Autonomous Systems*, 54(3):244–255, Mar. 2006.

[39] Microsoft Corporation. Microsoft Surface. *http://www.microsoft.com/surface/*, accessed 19/03/2008.

[40] Microsoft Corporation. Microsoft Launches New Product Category: Surface Computing Comes to Life in Restaurants, Hotels, Retail Locations and Casino Resorts. Press Release, 29 May 2007. *http://www.microsoft.com/presspass/press/2007/may07/05-29MSSurfacePR.mspx*, accessed 19/03/2008.

[41] J. Miura and Y. Shirai. Parallel Scheduling of Planning and Action of a Mobile Robot Based on Planning-Action Consistency. In *Proceedings of the IJCAI-99 Workshop on Scheduling meet Real-time Monitoring in a Dynamic and Uncertain World; Stockholm, Sweden*, 1999.

[42] T. Miyashita, T. Tajika, H. Ishiguro, K. Kogure, and N. Hagita. Haptic Communication between Humans and Robots. In *12th International Symposium of Robotics Research*, 2005.

[43] N. Muscettola, G. A. Dorais, C. Fry, R. Levinson, and C. Plaunt. IDEA: Planning at the Core of Autonomous Reactive Agents. In *Proceedings of the Workshops at the AIPS-2002 Conference, Toulouse*, 2002.

[44] K. Nickel, H. K. Ekenel, M. Voit, and R. Stiefelhagen. Audio-Visual Perception of Humans for a Humanoid Robot. In *Proceedings of the 2nd International Workshop on Human-Centered Robotic Systems (HCRS)*, München, Germany, 2006.

[45] S. Oviatt. Ten Myths of Multimodal Interaction. *Communications of the ACM*, 42(11):74–81, 1999.

[46] P. Rößler and U. D. Hanebeck. Telepresence Techniques for Exception Handling in Household Robots. In *Proceedings of the 2004 IEEE Intl. Conference on Systems, Man, and Cybernetics (SMC'04)*, Den Haag, Netherlands, 2004.

[47] P. Scattolin. Recognition of handwritten numerals using elastic matching. Master's thesis, Concordia University, Montréal, Canada, 1995.

[48] A. J. Schmid, M. Hoffmann, and H. Wörn. A Tactile Language for Intuitive Human-Robot Communication. In *Proceedings of the 9th International Conference on Multimodal Interfaces (ICMI 2007)*, Nagoya, Japan, 2007.

[49] A. J. Schmid, M. Hoffmann, and H. Wörn. A Tactile Language for Intuitive Human-Robot Communication. In *Proceedings of the 2007 IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, 2007.

[50] A. J. Schmid, O. C. Schrempf, U. D. Hanebeck, and H. Wörn. A Novel Approach To Proactive Human-Robot Cooperation. In *Proceedings of the 14th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN 2005)*, Nashville, TN, 2005.

[51] A. J. Schmid, O. Weede, and H. Wörn. Proactive Robot Task Selection Given a Human Intention Estimate. In *Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication 2007 (RO-MAN 2007)*, Jeju Island, South Korea, 2007.

[52] K.-U. Scholl. Modular Controller Architecture. *http://mca2.org*, accessed 14/04/2008.

[53] K.-U. Scholl, J. Albiez, and B. Gassmann. MCA - An Expandable Modular Controller Architecture. *3rd Real-Time Linux Workshop*, 2001.

[54] O. C. Schrempf and U. D. Hanebeck. A Generic Model for Estimating User Intentions in Human-Robot Cooperation. In *Proceedings of the* 2^{nd} *International Conference on Informatics in Control, Automation and Robotics, ICINCO 05*, 2005.

[55] J. Serra. *Image Analysis and Mathematical Morphology*. Academic Press, London, 1982.

[56] R. Simmons, J. Fernandez, R. Goodwin, S. Koenig, and J. O'Sullivan. Xavier: An Autonomous Mobile Robot on the Web. *Robotics and Automation Magazine*, 1999.

[57] R. Simmons, D. Goldberg, A. Goode, M. Montemerlo, N. Roy, B. Sellner, C. Urmson, A. Schultz, M. Abramson, W. Adams, A. Atrash, M. Bugajska, M. Coblenz, M. MacMahon, D. Perzanowski, I. Horswill, R. Zubek, D. Kortenkamp, B. Wolfe, T. Milam, and B. Maxwell. GRACE: An Autonomous Robot for the AAAI Robot Challenge. *AI Mag.*, 24(2):51–72, 2003.

[58] R. G. Simmons. Structured Control for Autonomous Robots. *IEEE Transactions on Robotics and Automation*, 10(1), Feb. 1994.

[59] M. Simon. *Automata Theory*. World Scientific, 1999.

[60] J. Sklansky and V. Gonzalez. Fast polygonal approximation of digitized curves. *Pattern Recognition*, 12(5):327–331, 1980.

[61] R. Stiefelhagen, C. Fügen, P. Gieselmann, H. Holzapfel, K. Nickel, and A. Waibel. Natural human-robot interaction using speech, head pose and gestures. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2004)*, Sendai, Japan, 2004.

[62] G. Strube. Modelling Motivation and Action Control in Cognitive Systems. *Mind Modelling. Berlin: Pabst*, pages 111–130, 1998.

[63] S. Thrun, M. Bennewitz, W. Burgard, A. B. Cremers, F. Dellaert, D. Fox, D. Hähnel, C. Rosenberg, N. Roy, J. Schulte, and D. Schulz. MINERVA: A Second-Generation Museum Tour-Guide Robot. In *Proceedings of the 1999 IEEE International Conference on Robotics and Automation*, 1999.

[64] T. Tyrrell. *Computational Mechanisms for Action Selection*. PhD thesis, University of Edinburgh, Edinburgh, Scotland, 1993.

[65] K. Wada, T. Shibata, T. Saito, and K. Tanie. Robot Assisted Activity for Elderly People and Nurses at a Day Service Center. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 2, pages 1416–1421 vol.2, 2002.

[66] K. Wada, T. Shibata, T. Saito, and K. Tanie. Effects of Robot-Assisted Activity for Elderly People and Nurses at a Day Service Center. *Proceedings of the IEEE*, 92(11):1780–1788, 2004.

[67] O. Weede. Proaktives Handeln für einen humanoiden Roboter. Diplomarbeit, Universität Karlsruhe (TH), Germany, 2006.

[68] K. Weiß. Weiss Robotics. *http://www.weiss-robotics.de*, accessed 19/03/2008.

[69] W. Westerman. *Hand Tracking, Finger Identification, and Chordic Manipulation on a Multi-Touch Surface*. PhD thesis, University of Delaware, 1999.

[70] T. Wösch and W. Feiten. Reactive Motion Control for Human-Robot Tactile Interaction. In *2002 IEEE International Conference on Robotics & Automation*, Washington, DC, May 2002.

[71] J. Yen, J. Yin, T. Ioerger, M. Miller, D. Xu, and R. Volz. CAST: Collaborative Agents for Simulating Teamwork. In *Proceedings of the 7th International Joint Conference on Artificial Intelligence (IJCAI-01)*, volume 17, pages 1135–1144, 2001.

Publications of the Author

A. J. Schmid, N. Gorges, D. Göger, and H. Wörn. Opening a Door with a Humanoid Robot Using Multi-Sensory Tactile Feedback. In *Proceedings of the 2008 IEEE International Conference on Robotics and Automation*, Pasadena, CA, 2008.

A. J. Schmid, M. Hoffmann, and H. Wörn. A Tactile Language for Intuitive Human-Robot Communication. In *Proceedings of the 2007 IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, 2007.

N. Gorges, A. J. Schmid, D. Osswald, and H. Wörn. A Framework for Creating, Coordinating, and Executing Skills on a Humanoid Robot. In *Proceedings of the 2007 IEEE-RAS International Conference on Humanoid Robots*, Pittsburgh, PA, 2007.

C. Burghart, R. Mikut, T. Asfour, A. J. Schmid, F. Kraft, O. Schrempf, H. Holzapfel, A. Swerdlow, G. Bretthauer, and R. Dillmann. Kognitive Architekturen für humanoide Roboter: Anforderungen, Überblick und Vergleich. In *Proceedings 17. Workshop Computational Intelligence*, Dortmund, Deutschland, 2007. Universitätsverlag Karlsruhe.

A. J. Schmid, M. Hoffmann, and H. Wörn. A Tactile Language for Intuitive Human-Robot Communication. In *Proceedings of the 9th International Conference on Multimodal Interfaces (ICMI 2007)*, Nagoya, Japan, 2007.

A. J. Schmid, O. Weede, and H. Wörn. Proactive Robot Task Selection Given a Human Intention Estimate. In *Proceedings of the 16th IEEE International Symposium on Robot and Human Interactive Communication 2007 (RO-MAN 2007)*, Jeju Island, South Korea, 2007.

A. J. Schmid, N. Gorges, and H. Wörn. Towards tightly-coupled robot-human interaction. In *Proceedings of the Workshop VII - Physical Human-Robot Interaction in Anthropic Domains held in conjunction with the 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'06)*, Beijing, China, 2006.

A. J. Schmid, H. Wörn, O. C. Schrempf, and U. D. Hanebeck. Towards Intuitive Human-Robot Cooperation. In *Proceedings of the 2nd International Workshop on Human-Centered Robotic Systems (HCRS)*, München, Germany, 2006.

A. J. Schmid, O. C. Schrempf, U. D. Hanebeck, and H. Wörn. A Novel Approach To Proactive Human-Robot Cooperation. In *Proceedings of the 14th IEEE International Workshop on Robot and Human Interactive Communication (RO-MAN 2005)*, Nashville, TN, 2005.

A. J. Schmid and H. Wörn. Path Planning for a Humanoid Robot Using NURBS Curves. In *Proceedings of the 2005 IEEE Conference on Automation Science and Engineering (IEEE CASE 2005)*, Edmonton, Canada, 2005.

A. J. Schmid. Implementierung eines PROFIBUS-DP-Controllers mit VHDL. Diplomarbeit, Universität Stuttgart, Germany, 1999.

A. J. Schmid. Network Monitoring with Focus on HTTP. Master's thesis, Oregon State University, USA, 1998.

A. J. Schmid. Untersuchung des Elektroklinen Effektes bei SmA-Testzellen. Studienarbeit, Universität Stuttgart, Germany, 1995.