

Adaptive Task Allocation, Localization and Motion Planning for the Multi-Robot System

Dali Sun

Dissertation zur Erlangung des akademischen Grades Doktor der Naturwissenschaften
Technische Fakultät, Albert-Ludwigs-Universität Freiburg

Dekan	Prof. Dr. Oliver Paul
Erstgutachter	Prof. Dr. Bernhard Nebel Albert-Ludwigs-Universität Freiburg
Zweitgutachter	Prof. Dr. Wolfram Burgard Albert-Ludwigs-Universität Freiburg
Tag der Disputation	21. September 2017

Abstract

With shortening product life cycles and increasing demands for product diversifications, a fully autonomous transport system with high performance and flexibility is strongly expected. The relevant study becomes more and more attractive and important. In this thesis, we focus on three important problems involved with it.

The first problem is the Multi-Robot Task Allocation (MRTA) problem. In this thesis, a novel approach named Distributed Heterogeneous Hash Tables (DHHT) is proposed to solve the task allocation problem. Our approach assigns robots to a set of transportation tasks offered by loading stations in two steps: firstly, a robot is assigned to the station; then the robot gets a specific transportation task from the station with the help of a contract net protocol. The experimental results have shown that our approach reached a fair distribution of robots among stations, and thus achieved a high total delivery efficiency of the system.

The second problem is the localization of the autonomous mobile robot. We introduce an effective and robust dynamic localization method, which separates localization from the mapping process. In localization process, a particle filter coupled with a distance filter and a scan matching method achieves a more robust localization against dynamic obstacles. In mapping process, we introduce an extend sensor model which takes dynamic influences as well as the state transition probability of the map directly into the map update step. The empirical evaluation shows that our approach achieves a robust localization in several highly dynamic real-world environments. At the same time the mapping stays consistent with the ground truth.

The third relevant problem is the Multi-Robot Motion Planning (MRMP) problem. At first, we propose a road map based approach called Adaptive Road Map Optimization (ARMO) to deal with the Multi-Robot Path Planning (MRPP) problem. Our approach generates an optimal road map by considering environment constraints and the current demands of stations in a pre-processing phase. Then, the robots extract individual paths from the optimal road map by a local path planner and navigate themselves cooperatively by a local navigation module. The experimental results showed that the ARMO approach lead to a more efficient navigation than conventional prioritized based decoupled path planning (PRIOR) methods. Furthermore, we propose a Behavior-Based Multi-Robot Collision Avoidance (BBMRCA) method inspired by swarm intelligence to solve the multi-robot motion planning problem. We evaluated BBMRCA methods with up to 100 robots in a simulation environment and the experimental results have shown

that the quality of solutions and the scalability are at the same level as ARMO and significantly better than those of conventional decoupled prioritized planning methods.

The work presented in this thesis has been conducted within the context of the project “KARIS (*Kleinskalige Autonomes Redundantes Intralogistiksystem*)” [42] and its follow-up project “KARIS Pro” [83]. The projects act as a focal point for the research and as a result that provide a distributed multi-robot system KARIS to accomplish transportation tasks in the field of intra-logistics and production. So in the end, we describe a demonstration and two pilot applications within the two projects roughly.

In summary, we propose a fully decentralized task allocation method which can facilitate a fair distribution of robots among the loading stations. We introduce an effective and robust localization method to achieve an accurate localization in highly dynamic real world environments. We also propose two decoupled multi-robot motion planning methods which can efficiently navigate a huge number of robots in a structured and crowded environment. Through the experiments in the simulation and on the real system, we have shown that our system is able to self-organize the transport task in a human workspace in a more flexible and efficient way.

Parts of this thesis have been published in the following papers:

- Dali Sun, Alexander Kleiner und and C. Schindelhauer. **Decentralized Hash Tables For Mobile Robot Teams Solving Intra-Logistics Tasks.** In Proceedings of the 9th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010), S. 923-930. 2010.
- Alexander Kleiner, Dali Sun und D. Meyer-Delius. **ARMO: Adaptive Road Map Optimization for Large Robot Teams.** In Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS). 2011.
- Dali Sun, Alexander Kleiner and Bernhard Nebel. **Behavior-based Multi-Robot Collision Avoidance.** In Proceedings of the IEEE International Conference on Robotics and Automation (ICRA). 2014.
- Dali Sun, Florian Geißer and Bernhard Nebel. **Towards Effective Localization in Dynamic Environments.** In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2016). 2016.

Published papers that are not part of this thesis:

- Christian Becker-Asano, Dali Sun, Birgit Kleim, Corinna N. Scheel, Brunna Tuschen-Caffier und Bernhard Nebel. **CoVE: Coping in Virtual Emergencies.** In Workshop on Emotion and Computing - Current Research and Future Impact, S. 1. 2011.
- Christian Becker-Asano, Dali Sun, Birgit Kleim, Corinna Scheel, Brunna Tuschen-Caffier und Bernhard Nebel. **Outline of an Empirical Study on the Effects of Emotions on Strategic Behavior in Virtual Emergencies.** In Affective Computing and Intelligent Interaction, S. 508-517. 2011.
- Christian Becker-Asano, Dali Sun, Corinna N. Scheel, Brunna Tuschen-Caffier und Bernhard Nebel. **Analyzing for emotional arousal in HMD-based head movements during a virtual emergency.** In Intl. Workshop on Emotion and Computing in conj. with KI2013. 2013.
- Corinna N. Scheel, Birgit Kleim, Julian Schmitz, Christian Becker-Asano, Dali Sun, Bernhard Nebel und Brunna Tuschen-Caffier. **Psychophysiologische Belastungsreaktivität nach einem simulierten Feuer in einer Parkgarage.** Zeitschrift für Klinische Psychologie und Psychotherapie 41 (3), S. 180-189. 2012.

Zusammenfassung

Durch den verkürzten Lebenszyklus und die steigende Variantenvielfalt von Produkten, ist die Anforderung an ein vollautonomes, flexibles und effizientes Transportsystem gestiegen. Mehr denn je sind damit Forschungen in diesem Gebiet wichtig und attraktiv. In dieser Arbeit konzentrieren wir uns auf drei relevante Probleme in diesem Bereich.

Das erste Problem ist die Multi-Roboter-Aufgabenverteilung. In dieser Arbeit wird ein neuartiger Ansatz “Distributed Heterogene Hash Tables (DHHT)” vorgestellt. Mit unserem Ansatz werden Transportaufgaben von Stationen den Robotern in zwei Schritten zugeteilt: Im ersten Schritt wird einem Roboter eine Station zugeordnet; danach bekommt der Roboter eine spezifische Transportaufgabe von der Station mit Hilfe eines sogenannten Contract-Net-Protokolls zugewiesen. Die experimentellen Ergebnisse haben gezeigt, dass unser Ansatz eine gerechte Verteilung von Robotern auf Stationen erreichen kann und somit eine hohe Effizienz des gesamten Systems erzielt.

Als zweites Problem untersuchen wir das Lokalisierungsproblem. In dieser Arbeit wird eine neue effektive und robuste Lokalisierungsmethode für dynamische Umgebungen eingeführt. Unsere Methode zerlegt das Problem in zwei Teilprobleme: Lokalisierung und Mapping. Im Lokalisierungsprozess wird eine Monte-Carlo Lokalisierung zusammen mit einem modernem Scan-Matching Verfahren und einem Distanzfilter eingesetzt, um somit eine stabile Lokalisierung in der dynamischen Umgebung zu erzielen. Im Mappingprozess wird ein erweitertes Sensor-Modell eingeführt, um dynamische Einflüsse direkt in den Kartenaktualisierungsschritt zu integrieren. Die experimentellen Ergebnisse haben gezeigt, dass unser Ansatz mit Hilfe der aktualisierten Karte eine akkurate Positionsbestimmung in äußerst dynamischen Umgebungen gewährleisten kann.

Das dritte Problem ist das Multi-Roboter Navigationsproblem. Wir stellen zuerst einen Roadmap-basierten Ansatz vor: “Adaptive Road Map Optimierung (ARMO)”. Dieser berechnet in einer Vorverarbeitungsphase eine optimale Roadmap unter Berücksichtigung der eingeschränkten Umgebung und aktuellen Anforderungen von Stationen. Anschließend extrahiert jeder Roboter seinen individuellen Pfad in der Roadmap mit Hilfe eines lokalen Pfadplaners, und führt diesen Pfad aus. Die experimentellen Ergebnisse haben gezeigt, dass dieser Ansatz weit effizienter als ein entkoppeltes und priorisierendes Multi-Roboter Pfadplanungsverfahren ist.

Darüber hinaus stellen wir außerdem eine Methode zur Multi-Roboter Kollisionsvermeidung vor. Diese basiert auf dem Prinzip der Schwarmintelligenz, wird jedoch zusätzlich mit einer entkoppelten und priorisierenden Pfadplanungsmethode kombiniert, um Kollisionen der Roboter untereinander zu vermeiden. Wir bewerten unsere Methoden mit bis zu 100 Robotern in einer Simulationsumgebung. Die Ergebnisse zeigen, dass die Effizienz und Skalierbarkeit unserer Methode auf dem gleichen Niveau der Roadmap-basierten Methode ist, und sie deutlich besser ist als konventionelle entkoppelte Multi-Roboter Pfadplanungsmethoden.

Die vorgestellte Arbeit wurde im Rahmen des Projekts “KARIS (*Kleinskaliges Autonomes Redundantes Intralogistiksystem*)” [42] und des Folgeprojekts “KARIS Pro” [83] erstellt. Beide Projekte setzen den Schwerpunkt auf Forschung, Entwicklung und Implementierung eines autonomen Multi-Roboter Transportsystems im Bereich der Intralogistik und Produktion. Die Arbeit schließt mit einer Demonstration zweier Pilotanwendungen, die innerhalb dieser Projekte entstanden sind.

Zusammengefasst haben wir eine dezentralisierte Methode zur Aufgabenverteilung vorgestellt, die die gerechte Verteilung der Roboter auf Stationen begünstigen kann. Wir haben zusätzlich eine effektive und robuste Lokalisierungsmethode entwickelt, die eine akkurate Lokalisierung in hoch-dynamischen Umgebungen erzielen kann. Wir haben zwei dezentralisierte Multi-Roboter Navigationsverfahren eingeführt, die eine reibungslose Navigation einer großen Anzahl von Robotern in einer strukturierten und eingeschränkten Umgebung gewährleisten kann. Durch die Experimente in der Simulation und auf dem realen System haben wir gezeigt, dass Transportaufgaben in einer komplizierten Arbeitsumgebung von unserem System selbstständig, flexibel und effizient durchgeführt werden können.

Acknowledgements

First of all I would like to thank Prof. Bernhard Nebel for supporting me over all these years. I have learned a lot of things during the studies in our research group headed by him. Prof. Nebel is competent, conscientious and critical, and it was a great pleasure and honour to work in this group with him as my supervisor.

I would also like to thank prof. Wolfram Burgard for being my very competent co-examiner.

I would like to thank prof. Christian Schindelhauer, prof. Wolfram Burgard and prof. Kai Furmans for their advice and support for the KARIS project.

My special thanks goes to Alexandar Kleiner for his coming up with many interesting ideas that influenced this thesis. He spent a lot of time in helping me to solve soft- and hardware problems and to carry out the experiments.

Furthermore, I would like to thank Daniel Meyer-Delious, Florian Geisser, Thomas Stoll and Andreas Trenkle for the great collaboration during the KARIS and KARIS Pro project. It was a pleasant experience to work with them in both projects.

Besides, I would like to thank all my colleagues in our research group for the pleasant atmosphere. I have learned a lot from my colleagues through many fruitful discussions. Many of them have given me a number of interesting ideas and helpful suggestions for this thesis. This thesis could not have arrived at final success without the help and support from them. Especially, I would like to thank:

Christian Becker-Asano for helping and supporting me during the studies in FRIAS, and for his beneficial suggestions for improving my research studies,

Christian Dornhege for many useful suggestions for improving the software quality and for helping me to solve several soft- and hardware problems,

Uli Jakob for helping and supporting me by solving many technical problems,

Roswitha Hilden and Petra Geiger for helping me on administrative matters.

Moreover, I would like to acknowledge the collaborative work with the Institute for Material Handling and Logistics, Karlsruhe Institute of Technology, and all partner companies involved in the KARIS and KARIS Pro projects.

Finally, I would like to thank my parents who have always supported me for my longtime abroad studies and my wife Wenjuan for taking care the family, cooking delicious Chinese food, and enjoying life with me.

Contents

Abstract	iii
Zusammenfassung	vii
Acknowledgements	ix
Contents	xi
List of Figures	xiii
List of Tables	xvii

1	Introduction	1
2	Distributed Multi-robot Task Assignment Problem	5
2.1	Introduction	5
2.2	Related Work	9
2.3	Distributed Multi-robot Task Allocation	11
2.3.1	Distributed Heterogeneous Hash Tables	12
2.3.1.1	Station Status Information and Weight	15
2.3.2	Mobile Content-Addressable Network	16
2.3.2.1	Geographic Network Routing	17
2.3.2.2	MCAN Construction and Maintenance	18
2.3.3	DHHT-based Task Allocation	19
2.4	Experiments and Results	20
2.4.1	Simulation Results	21
2.4.1.1	Efficiency and Convergence	23
2.4.1.2	Adaption to sudden changes	24
2.4.2	Real-World Results	26
2.5	Conclusion	30
3	Localization in Dynamic Environments	31
3.1	Introduction	31
3.2	Related Work	33
3.3	Dynamic Localization	35

3.3.1	Distance Filter	36
3.4	Dynamic Mapping	37
3.5	Experimental Results	40
3.5.1	Localization results	44
3.5.2	Mapping results	45
3.6	Conclusion	47
4	Adaptive Road Map Optimization	51
4.1	Introduction	51
4.2	Related Work	54
4.3	Adaptive Road-Map Generation and Optimization	57
4.3.1	Detection and Integration of Dynamic Changes	59
4.3.2	Computation of the Connectivity Network	59
4.3.3	Definition of the LP Problem	60
4.3.4	Local Planner	62
4.4	Experiments and Results	64
4.4.1	Adaption in Dynamic Environments	67
4.4.2	Real-World Experiments and Results	68
4.5	Conclusion	72
5	Swarm-Intelligence: Behavior-based Collision Avoidance	73
5.1	Introduction	73
5.2	Related Work	76
5.3	Behavior-based Collision Avoidance	80
5.3.1	Problem Formulation	80
5.3.2	Swarm Behaviors and Traffic rules	81
5.3.2.1	Definition of behaviors	83
5.3.2.2	Definition of the Traffic rules	85
5.3.3	Deadlock and Re-plan	86
5.4	Experimental Results	90
5.5	Conclusion	93
6	The KARIS System	95
6.1	Introduction	95
6.2	The KARIS Project	95
6.2.1	Application: A Real World Demonstration	97
6.3	The KARIS Pro Project	99
6.3.1	Two Pilot Applications	100
7	Conclusions	103
	Bibliography	107

List of Figures

2.1	(a) Black dotted lines denote the direct connections between robots in the MCAN. (b) Routing along the red dotted lines indicate the courses from a mobile robot to a stationary robot (station).	18
2.2	Visualization of $\min(D_i)$ for two different situations on the <i>small map</i> . Weighted distances D_i of each station i are depicted by a specific color. For each location the minimum D_i is drawn.	20
2.3	(a,c) Simulated environments in USARSim. (d,e) Generated grid maps for localization and planning with superimposed Voronoi graph (red). Maps are denoted as (a) <i>Small factory</i> , (c) <i>Big factory</i> and (e) <i>Company map</i>	22
2.4	Efficiency of all 12 stations over time on the <i>Small map</i> . Each color denotes a single station. (a) Baseline approach, and (b) DHHT approach.	24
2.5	Efficiency of all 8 stations over time on the <i>Company map</i> . Each color denotes a single station. (a) Baseline approach, and (b) DHHT approach.	25
2.6	Adaption after rapid throughput change. For the first 30 minutes all 12 station were running at $\lambda = 8$, and from then on 6 with $\lambda = 10$ and 6 with $\lambda = 4$	26
2.7	Adaption after a sudden break-down. For the first 30 minutes the system was running with 18 robots, then, we intentionally crashed 10 of them.	27
2.8	The second prototype of KARIS robot.	28
2.9	Map of the real-world scenario and visualization of $\min(D_i)$. The far-off station on the left side (dark orange) has the highest demand for robots.	28
2.10	Efficiency of all 3 stations over time in the real environment. (a) Baseline approach, and (b) DHHT approach.	29
3.1	Bayesian network of the dynamic localization problem.	36
3.2	Hidden Markov Model of the extended sensor model.	39
3.3	Ground truth maps of collected data sets of PH1.	41
3.4	Ground truth maps of collected data sets of PH2.	42
3.5	Ground truth maps of collected data sets of LC.	43
3.6	Mapping and localization results for PH1	47
3.7	Mapping and localization results for PH2	47
3.8	Mapping and localization results for LC	48

4.1	System Overview	58
4.2	In industrial environments the map can locally change due to replaced objects, such as pallets, as well as gathering humans. Adaptive road map optimization facilitates the simultaneous navigation planning of large robot teams when taking these changes into account.	60
4.3	Grid maps utilized for experiments: (a) the PLANT map generated from a simulated environment, (b) the ASE map generated in a real logistic center, (c) the KNO map generated in a large distribution center.	65
4.4	Comparing the CTime of prioritized planning (PRIOR) with adaptive road map optimization (ARMO).	65
4.5	Adjustment of the road map according to dynamic changes in the map (a,c) source of disturbance and (b,d) resulting modifications reflected on the road map.	67
4.6	Real world experiments in the robot experimental area of University of Freiburg. (a)Snapshot pictures captured from Stage. (b) and (c) show four robots and the environment during the experiments.	69
4.7	Standard deviation of fairness over time. Each color denotes a pair of methods.	70
4.8	Efficiency of 3 stations over time. Each color denotes a single station. Station 4 has no outgoing tasks.	71
5.1	The swarm robots of project SYMBRION form a 3D structure to collectively pass a barrier (http://zool33.uni-graz.at/artlife/SYMBRION).	79
5.2	A Low Cost Scalable Robot System Kilobot (http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html).	79
5.3	The Swarmanoid robot system consists of three different robot types: (a) foot-bots, (b) hand-bots and (c) eye-bots (http://www.swarmanoid.org/swarmanoid_hardware.php).	80
5.4	Definition of the front area and the critical area.	82
5.5	Snapshot pictures captured from Stage utilized for experiments: (a) the <i>Building_78</i> map generated in robot experimental area of University of Freiburg, (b) the <i>ASE</i> map generated in a real logistic center. (c) <i>KNO</i> map generated in a large distribution center.	91
6.1	(a) The third prototype of KARIS robot. (b) Inter-robot connections facilitated by a docking mechanism for jointly solving tasks.	96
6.2	The abstract functional software architecture.	97
6.3	The demonstration at the <i>Logimat</i> Fair. (a) The scenario of the demonstration and the red cube denotes a KARIS robot. (b) A robot is docked with a station and is loading a crate. (c) A red crate used to transport the foods and coffees. (d) A robot is approaching a recharging position. (e) The safe navigation among humans. (f) Four KARIS robots used in this Demonstration.	98

6.4	The multi-company research project KARIS Pro with our institute taking the responsibility for localization and navigation.	99
6.5	(a) A robot equipped with a module for small load carrier is docking with a loading station. (b) Robots are charging with contact-less charger stations (c) A robot docks with a cart and moves together with it.	100

List of Tables

2.1	Simulation Results DHHT & Baseline	23
2.2	Real-World Results DHHT & Baseline	29
3.1	Learned parameters of the extended sensor model, denoted in %. . .	44
3.2	Description of static and dynamic methods	45
3.3	Comparison results of static and dynamic methods	46
4.1	Comparing prioritized planning (PRIOR) with adaptive road map optimization (ARMO)	66
4.2	Results of Real-World experiments of map Building_78	70
5.1	Experiment results of map Building_78.	92
5.2	Experiment results of map ASE.	92
5.3	Experiment results of map KNO.	93

For my family and friends

Chapter 1

Introduction

With shortening product life cycles and increasing demands for product diversifications, flexibility and re-configurability are increasingly needed in the field of logistics and production. However, fixed conveyor systems and human labor are still widely used for flow of goods. The main drawbacks are the lack of flexibility, high labor cost and risk of injury. Some commercial systems such as the TransCar system [105] and the Kiva system [119] are already running in hospitals and warehouses. However, they are implemented by a centralized control and some require the user to engineer the environment for human-free workspaces. Therefore, a fully autonomous transport system with high performance and flexibility is strongly expected to overcome the drawbacks and thus the relevant study becomes more and more attractive and important. In this thesis, we focus on three important problems involved with it.

The first important problem is the Multi-Robot Task Allocation (MRTA) problem, namely, to decide which task should be taken by which robot and when it should happen. In this thesis, a novel approach named Distributed Heterogeneous Hash Tables (DHHT) is proposed to solve the task allocation problem. Our approach assigns robots to a set of transportation tasks offered by loading stations in two steps: firstly, a robot is assigned to the station with the DHHT method; then the robot gets a specific transportation task from the station with the help of a contract net protocol. The robots build a kind of Mobile Content-Addressable Network (MCAN) in which all messages are delivered via a so-called geographic routing scheme. Our approach is completely decentralized, i.e., it needs no central server to coordinate the task allocation. Only robots and loading stations carry communication devices and all necessary communications are performed by the robots and the loading stations. Therefore, our approach can be used in many

production plant environments without installing additional hardware infrastructures. This approach was evaluated in both simulation and real world scenarios. The experimental results have shown that our approach reached a fair distribution of robots among stations under several different test constraints, and thus achieved a high total delivery efficiency of the system.

The second problem is the localization of the autonomous mobile robot, namely, to keep track of a robot's location within a map built for the environment. While common approaches are successfully applied in static environments, they may not be well-suited for highly dynamic and complex environments. The reason is that changes in such environments occur consistently over time and the localization methods, which have not take dynamic influences into account, may fail after a long time. We introduce an effective and robust dynamic localization method, which separates localization from the mapping process. In localization process, a particle filter coupled with a distance filter and a scan matching method achieves a more robust localization against dynamic obstacles. In mapping process, we introduce an extend sensor model which takes dynamic influences as well as the state transition probability of the map directly into the map update step and leads to a more accurate and robust map update. The empirical evaluation shows that our approach achieves a robust localization in several highly dynamic real-world environments, all including dynamic influences. At the same time the mapping stays consistent with the ground truth.

The third relevant problem is the Multi-Robot Motion Planning (MRMP) problem, namely, to find out how a team of autonomous mobile robots can move in the same environment at the same time while avoiding collisions with other robots and obstacles. At first, we propose a road map based approach called Adaptive Road Map Optimization (ARMO) to deal with the Multi-Robot Path Planning (MRPP) problem. Our approach generates an optimal road map by considering environment constraints and the current demands of stations in a pre-processing phase. Then, the robots extract individual paths from the optimal road map by a local path planner and navigate themselves cooperatively by a local navigation module. The optimal road map will be recomputed only when changes of environments or stations have been detected. Furthermore, the computation time of the generation of an optimal road map depends mainly on the complexity of the environment rather than on the number of robots. Therefore, our approach is able to be used for motion planning of a large robot team (up to 100 robots) in dynamically changing environments. We conducted several simulation experiments to test our approach under difficult conditions. The experimental results showed

that the ARMO approach lead to a more efficient navigation than conventional prioritized based decoupled path planning (PRIOR) methods.

Furthermore, we propose a Behavior-Based Multi-Robot Collision Avoidance (BBMRCA) method inspired by swarm intelligence to solve the multi-robot motion planning problem in a totally different way. We combined the advantages of the swarm techniques and the decoupled multi-robot path planning method in order to achieve higher flexibility and better quality. We evaluated BBMRCA methods with up to 100 robots in a simulation environment and the experimental results have shown that the quality of solutions and the scalability are at the same level as ARMO and significantly better than those of conventional decoupled prioritized planning methods. The main advantage of our approach compared with ARMO is that BBMRCA doesn't need any pre-processing such as computing the optimal road-map, which requires much time for large and complex environments. Our method is as flexible as PRIOR, when an unexpected and unknown obstacle appears. At the same time, compared with common swarm technologies, our method is more suitable in complicated, structured and crowded environments, such as confined and cluttered logistic centers and manufacturing plants.

The work presented in this thesis has been conducted within the context of the project “KARIS (*Kleinskalige Autonomes Redundantes Intralogistiksystem*)” [42] and its follow-up project “KARIS Pro” [83]. The projects act as a focal point for the research and as a result that provide a distributed multi-robot system KARIS to accomplish transportation tasks in the field of intra-logistics and production. So in the end, we describe a demonstration and two pilot applications within the two projects roughly. The demonstration was carried out at the *Logimat* Fair in Stuttgart 2010. We have successfully shown the advantages of the system against conventional transport systems. The first pilot application was carried out for the Robert Bosch GmbH, Stuttgart, Germany, in the field of intra-logistics. The second pilot application was carried out for the quattro GmbH, Neckarsulm in Germany, in the field of production. Through two pilot applications, it has successfully been shown that the KARIS system is a realistic, flexible and efficient solution for the materials transport problems in the industrial-like domain.

In summary, we present methods for a distributed multi-robot system KARIS to solve the materials flow problems in industrial environments. We propose a fully decentralized task allocation method which can facilitate a fair distribution of robots among the loading stations. We introduce an effective and robust localization method to achieve an accurate localization in highly dynamic real world environments. We also propose two decoupled multi-robot motion planning methods which can efficiently navigate a huge number of robots in a structured and

crowded environment. Through the experiments in the simulation and on the real system, it has been shown that our system is able to self-organize the transport task in a human workspace in a more flexible and efficient way.

Chapter 2

Distributed Multi-robot Task Assignment Problem

2.1 Introduction

The task assignment problem is one of the fundamental combinatorial optimization problems in mathematics [13]. which can be described as follows: there are n agents and m tasks and any task can be performed by any agent. A cost or a profit is given to each pair of agent and task. The question is how to find an assignment of agents to tasks, so that all tasks can be completed and the total cost or profit is optimized.

In the classical task assignment problems the number of agents n is equal to the number of tasks m and an agent can only be mapped to exactly one task. In this case, the assignment can be treated as a bijective mapping between a set of agents and a set of tasks. If the tasks cannot be executed in parallel and the cost denotes the time needed to complete a task by a certain agent, the problem is called the linear sum assignment problem [13, Chapter 4] and the goal of such a problem is to find a mapping that minimizes the total completion time. If the tasks can be executed in parallel, the problem is called linear bottleneck assignment problem [13, Chapter 6]. The goal of a linear bottleneck assignment problem is to find a mapping that minimizes the latest completion time. The linear sum and bottleneck assignment problems are both called linear assignment problems, because the objective function (total cost or profit) to be optimized is a linear function. Applications which can be modeled as a linear problem are numerous and varied. For example, the personnel assignment problem can be modeled as a linear sum assignment problem. The problem of scheduling tasks in parallel

machines can be modeled as a linear bottleneck problem. Such a problem can be solved, for example, with the Hungarian method [48, Chapter 2] in $O(n^2)$ time.

The more complex problem with a quadratic objective function is called the quadratic assignment problem [12]. The quadratic problem can be formulated in the following way. N facilities must be allocated to n locations. A flow of materials or goods between facilities and a cost to place a facility at a certain location are given and the distance between arbitrary two locations is known. The goal is to find an assignment of facilities to locations so that the total cost, which consists of costs to place a facility at a location and costs depending on the flow between facilities multiplied by their distance, is minimized. In addition to the facility location problem described above, another interesting application of the quadratic assignment problem is the backboard wiring problem introduced by Steinberg [100]. There are different devices such as display, printer and computer which have to be placed in a room and connected to each other by wires. The problem is to place the devices in such a manner that the total wire length is minimized. Unlike the linear assignment problem which can be solved in polynomial time, this problem is a $\mathcal{NP}$ -hard problem [33] and there are no known methods with which such a problem can be solved efficiently.

When triples over 3 sets of equal size are considered instead of pairs from two sets, the assignment problem arises as a so-called 3-index assignment problem (3AP) [13, 99]. Such problems can be described as below. There are three n -sets A , B , and C , and to each triple in $(A \times B \times C)$, a weight is given either as a cost or as a profit. Depending on different requests for finding triples, the problem can be divided into two subclasses. One is called axial 3-index assignment problem (axial 3AP). In this subclass, we want to find n triples so that each element from A or B or C in n triples appears exactly only once and the cost or the profit is optimized. The other one is called planar 3-index assignment problem (planar 3AP). In this problem, we want to find n^2 triples so that each pair from $(A \times B)$ or from $(A \times C)$ or from $(B \times C)$ in n^2 triples appears exactly only once and the cost or profit is optimized. There are various applications which can be modeled as one of the 3APs. For example, the assembly of printed circuit boards problem [120] can be modeled as a axial 3AP. The capital investment problem [81] can be modeled as a axial 3AP. The most important applications of planar 3APs are in timetabling problems [41]. In the general case, the problem is called multi-index problem when the m -tuple instead of the triple is studied. The multi-index assignment problems (including 3APs) are also $\mathcal{NP}$ -hard.

In the context of multi-robot systems, the task assignment problem is known as Multi-Robot Task Allocation (MRTA) problem [35, 75]. In general, multi-robot task allocation problems are more complicated than classical assignment problems, because some extra factors must be handled, such as uncertainties, decentralized team organization and more complicated task setting. The study of multi-robot task allocation problems has become more and more important in the recent twenty years and works on it are widespread and varied. Most works on it have been focusing on the construction and validation in one or more application domains and varied in many ways, for instance, team size, team composition, communication topology and application domains.

The work of Gerkey et al. [35] presents a formal analysis of MRTA problems and provides a domain-independent characterization based on three axes:

1. Single-Task robots (ST) vs. Multi-Tasks robots (MT): ST and MT differ according to whether some robots can execute multiple tasks simultaneously or not.
2. Single-Robot tasks (SR) vs. Multi-Robot tasks (MR): SR and MR differ according to whether some tasks require multiple robots or not.
3. Instantaneous Assignment (IA) vs. Time-extended Assignment (TA): IA and TA differ according to whether tasks can be re-assigned or not. Tasks of IA can only be allocated once, compared to TA, the future assignment is available due to the lack of the information or because tasks will arrive over time.

A particular MRTA problem can be described by a triple of tree axes with their work. For example, the optimal assignment problem [31] is classified as an ST-SR-IA problem by using this taxonomy.

Although the categorizations in this work cover most of MRTR problems, there are still some problems that can not be included in this taxonomy, for example, the problems in which costs are dynamic and can vary during the runtime. Unfortunately the multi-robot materials flow problem that is dealt with in this thesis is an instance of such difficult problems. The travel cost of a certain station depends on how many other robots move to the same station at the same time, i.e., the travel cost is associated with the multi-robot motion planning problem. Obviously, such a problem is at least as difficult as a ST-SR-TA problem, which is a strongly $\mathcal{NP}$ -Hard problem [11].

There are several different approaches for solving MRTA problems, which range from simple greedy algorithms to auction-based approaches, emergent coordination techniques and fully distributed strategies.

The greedy algorithm [2] is the simplest algorithm widely used for task assignment problems as well as many other problems. The greedy algorithm is an iterative procedure which, at each iteration, takes the best choice at hand with the hope to find a solution that is good enough at the end of the procedure. But because greedy algorithms use a locally optimal strategy, the greedy algorithm based approaches often fail to find a global optimum.

Auction-based approaches [7, 19] inspired by sales auctions are another widely used method for task assignment problems. The basic idea of auction-based approaches is to build a price-based task market with a network protocol to find the highest bidder. The auction-based approach is firstly used for single task assignment problems. Many modifications and extensions are made to deal with multi-task problems. There are two main sub-categories of auction-based approaches: centralized and distributed. In centralized approaches, all tasks are assigned via a central unit. In distributed approaches, instead of a central unit, a consensus algorithm will be applied to assign tasks.

Biologically inspired emergent coordination approaches are very attractive to dynamic MRTA problems [63]. The approaches are based on local information and interactions, and individual robots coordinate their actions only through their interactions with each other or with the environment. There is no need for explicit negotiations between robots, and neither are direct communications necessary. “However, emergent systems tend not to be amenable to analysis, with their exact behavior difficult, if not impossible, to predict” (Gerkey et al. [35]).

Sub-grouping strategies [58] attempt to reduce the complexity of task allocation problems by subdividing problems into small subgroups. A global solution is made out of the proposed local solutions that are found in subgroups. In this approach, it is not easy to obtain a global optimal solution. Increasing the size of subgroup or restarting the searching at a specific decision circle will help solve the small to medium size problems. However, for large problems, to overcome local optimality is time-consuming and inefficient.

As mentioned above, the multi-robot materials flow problem described in this thesis is strongly $\mathcal{NP}$ -Hard. That means, there are even no known good approximation algorithms for such problems. Given the difficulty of the problems, we will

not attempt to solve it directly or exactly, but try to facilitate a even distribution of robots among stations in order to achieve a gut overall performance of the system.

The rest of this chapter is organized as follows: Section 2.2 reviews some related works about MRTA approaches. Section 2.3 describes our technique in detail. Section 2.4 demonstrates the experiments and results of our approach. Section 2.5 concludes our method and future work.

2.2 Related Work

Tovey et al. [109] present a decentralized auction-based approach for the multi-robot exploration task. Their research focuses on the systematic generation of bidding rules that determine, how a robot locally computes its bid so that a better performance for a particular global objective can be achieved. They demonstrate it by deriving bidding rules for three team objectives of a multi-robot exploration task: minimizing the total energy consumption, minimizing the task-completion time or minimizing the average target-visit time. Their experiments show a good performance compared to the optimal performance.

Zlot et al. [121] present a decentralized auction-based approach for multi-robot exploration task which applies multiple robots to collect relevant information from an unknown or partially known environment. At the beginning of exploration, each robot generates a list of goal points by using some strategies (Random, Greedy or Space Division) and plans a path to visit this goal. Next, during the exploration robots try to improve their plan via an auction mechanism continuously so that each goal will be assigned to a better-suited robot through negotiations. After the auction the goals in the list that a robot has are in most cases best-suited for this robot, namely, this robot can explore these goals most efficiently. Robots can also share information about which regions have already been visited or not. Their approach is completely distributed in that the exploration can still be carried out if some of robots lose communication or even fail completely. They have shown experimentally with a four-robot team that the exploration efficiency was improved by a factor of 3.4 by allowing the robots to negotiate using the auction mechanisms.

Choi et al. [64] present two decentralized consensus-based algorithms for single- and multi-task-assignment problems, respectively. The algorithms consist of iterations between two phases. The first phase is the auction process for decentralized

task selection by using greedy algorithms. The second phase is the consensus process for decentralized conflict resolutions from the first phase by using a consensus strategy. They introduce firstly a so-called consensus-based auction algorithm (CBAA) for the single-task-assignment problem and then extend it to a consensus-based bundle algorithm (CBBA) for the multi-task-assignment problem in which a sequence of multiple tasks is assigned to each robot. They demonstrate experimentally that their methods achieved good performance and quick convergence of the proposed methods compared with an existing sequential auction algorithm.

Gerkey et al. [34] present a decentralized auction-based allocation system called MURDOCH, using a simple auction protocol to allocate tasks and a resource centric, publish/subscribe messaging model for inter-robot communications. The resource centric, publish/subscribe messaging model is an instance of the anonymous communication. “A data producer simply tags a message with a subject describing its content and ‘publishes’ it onto the network; any data consumers who have registered interest in that subject by ‘subscribing’ will automatically receive the message. Data producers need not have any knowledge of which consumers, if any, are receiving their messages and vice versa [34, Section V].” The auctions protocol is a simple distributed contract net protocol that allocates tasks via a sequence of first-price auctions. They test and validate the MURDOCH approach in two domains: a loosely coupled multi-robot object-tracking task with 8 Pioneer 2-DX mobile robots and a tightly coupled multi-robot box pushing task with 3 Pioneer 2-DX mobile robots. They show empirically that such decentralized auction-based allocation mechanisms can be viable and effective for coordinating physical multi-robot systems.

Liu et al. [65] present a distributed auction-based approach for ST-SR-IA problems which produces a optimal solution with strongly polynomial time. In contrast to the conventional auction approaches in which a price escalation is encouraged by bidders, in this approach a price escalation is instigated by a so-called merchant who selects the pricing policy to steer purchasing behaviors of bidders. The computation and communication are distributed and can be decentralized and implemented for real robot systems without losing the global optimality. The time complexity of this method is $O(n^3 \lg(n))$ which is very close to the fastest centralized algorithms $O(n^3)$ where n is the number of the robots and in the case of ST-SR-IA problems, n is also the number of tasks.

2.3 Distributed Multi-robot Task Allocation

In this section, a novel approach to assign a set of mobile robots to a set of transportation tasks in an industrial environment will be introduced and analyzed. A transportation task is defined as the delivery of an object from a source station to a destination station. The objects will arrive at the different source stations over time and a mobile robot can only carry one object at once. Source and destination stations are placed in different locations in the environment and it is not necessary to have a global communication. Robots have only limited communication range and a mobile peer-to-peer network will be built to support communications between robots.

Robots are either mobile, which can move autonomously in the environment, or stationary, which will be fixed at some place and be used as a loading or unloading station. Both mobile and stationary robots have been equipped with a conveyor system which can load (unload) a crate automatically. Transportation tasks are expected to arrive over time among stations with different rates.

The goal is to find a dynamic assignment of mobile robots to tasks so that the average delivery efficiency of all stations during the entire transportation process can be optimized. But as is already mentioned in the introduction, the travel costs of deliveries depend also on the multi-robot motion planning problems and therefore are dynamic. The optimal solution in this situation is computationally intractable for large problems. Therefore, instead of searching a optimal solution, we try to distribute the robots among stations as evenly as possible. The rationale behind this idea is that on one hand the more transportation tasks a station has, the more robots this station needs; on other hand, the more robots are assigned to this station in the same period of time, the higher the travel cost to this station will be. Because the capacity of a road is limited, high traffic demand of robots will slow the speed of the traffic stream, especially in the vicinity of a station. In the worst case, it can result in some congestion. The first factor will increase the delivery efficiency of a station and the second factor will decrease the delivery efficiency of a station. Therefore, it is a good trade-off in most situations to facilitate a fair distribution of robots among stations. The formal definition of the delivery efficiency of a station will be given in the section 2.3.1.1.

In the following section, we will describe in depth how a even distribution can be achieved with the help of a so-called Distributed Heterogeneous Hash Tables

(DHHT). Furthermore, other key points, such as how to build a mobile peer-to-peer network and how to keep it intact, and how to maintain transportation tasks within the network, will be described in detail in this section.

2.3.1 Distributed Heterogeneous Hash Tables

The goal is to assign a set of robots $R_1, R_2, \dots, R_m$ to a set of stations $S_1, \dots, S_n$ as evenly as possible, which is very similar to storing a set of data elements to a set of slots. The hash table is a widely used data structure to store and to retrieve data efficiently. The main idea of the hash table is to use a so-called hash function to map an arbitrarily large domain to a small set of index values. Specifically, a hash function computes a so-called hash value from a search key of a given data element. This hash value is correspondent to an index of a slot and the data element then can be located directly with the computed hash value in the hash table. Note that in general, the domain of a hash function is larger than its range, e.g., the number of data elements is larger than the number of slots. So the mapping of keys into the hash range is normally not bijective, which means different keys can be mapped into the same hash value, i.e., different data elements can be stored in the same slot.

Since the hash value can be computed very efficiently, the hash table is a highly efficient data structure for a large set of data. But if the number of the slots varies (addition or removal of slots), the hash table needs to remap almost all keys to the slots, which also means that almost all data elements need to be moved. When such changes occur quite often, the hash table is rather inefficient. Karger et al. [52] introduce a so-called *consistent hashing* to overcome such drawbacks. They motivate the *consistent hashing* by data replication problems on the Internet: the server distributes objects among the caching machines to reduce the load, and clients use a hash function to retrieve a specific object from the caching machine. It happens very plausible on the Internet, that some caching machines can join or leave the network at any time.

Consistent hashing solves this problem in such a way that in addition to the mapping keys with a hash function, slots are also mapped to the same hash range by using an appropriate hash function. One hash function is used for slots to determine the boundaries of intervals and an other hash function is used for data to map the search key to the correct interval, namely corresponding slots. Note that these two hash functions are not necessarily identical, only the ranges of both functions must be identical. The range of hash functions is actually an

abstract keyspace, which is split by slots with a partitioning scheme. Each slot is responsible for a continuous interval in the keyspace, whereas in the conventional hash table, each slot is only correspondent to a single hash value. For example, the concept of *consistent hashing* can be visualized on a circle. Several slots partition the circle into several segments, with each slot is responsible one of these segments. An object is assigned to a slot, if the hash value of this object falls into the segment of this slot. With this property, when a slot is added, it takes over a part of the interval from the adjacent slots and builds its own interval. When a slot is removed, its interval is taken over by adjacent slots. In both cases, all other intervals remain untouched, thus only these data elements which are mapped to the changing intervals need to be reassigned to the corresponding slots. So *consistent hashing* can minimize the disruption when the size of hash table changes, and it allows the hash table to scale to extremely large numbers of slots and to handle continual slots arrivals and departures. This feature is called consistency.

In the naive version of *consistent hashing*, slots partition the keyspace normally not evenly, which can lead to a problem, that data is not evenly distributed among slots. Karger et al. proposed a solution in their work that each slot is represented by k virtual slots (copies) which are randomly located to multiple locations on the keyspace. They prove that when k is the logarithmic of the number of slots, the data elements are with high probability evenly distributed over all slots. This feature is called balance. Stoica et al. [101] have also shown this feature in their work “Chord”, when a sufficient number of copies of slots is used.

With *consistent hashing*, a change of slots causes only local maintenance of the hash table. This means that each slot does not need to have complete information about entire hash table, but just needs to maintain a set of links to its direct neighbors. These links together form the routing table which can be used to locate a certain slot. So the *consistent hashing* is mainly designed to solve Distributed Hash Table (DHT) problems. In the conventional DHT, slots are homogeneous and the different properties of slots, such as the size of storage and transmission bandwidth, are not considered. So Schindelhauer et al. [91] introduce a Weighted Distributed Hash Tables (WDHT) as an extension. They have deployed weighted consistent hashing to deal with the heterogeneous properties of slots. Every slot comes with a positive weight and a data element is assigned to a slot according to its relative weight. They present two methods which are called the Linear Method and the Logarithmic Method and proved that both of them can achieve a weighted balance.

The task assignment problem with which we are confronted in this chapter is strikingly similar to the WDHT. We want to assign robots to stations as evenly

as possible, while a station can join or leave at any time and the number of transportation tasks on each station can vary. So inspired by WDHT, we introduce so-called Distributed Heterogeneous Hash Tables (DHHT) to handle our assignment problem.

More formally, we assign a set of robots $R_1, R_2, \dots, R_m$ to a set of stations $S_1, \dots, S_n$ with weights $w_1, \dots, w_n$ under the constraints that balance and consistent properties of the WDHT are met. Moreover, considering the fact that the delivery time of a transportation task depends mainly on the distance between the robot and station, we add an extra constraint that a robot should be considered to be assigned to those stations which are close to it. We define this constraint as the locality property.

In DHHT, a weighted consistent hashing scheme is used to assign robots to stations. The keyspace is defined as a two-dimensional space $\mathcal{S}$ which is the area used by the robots. A relative simple hashing function maps robots and stations to their true positions in $\mathcal{S}$. The robots are uniformly randomly distributed in this space $\mathcal{S}$, whereas the stations are selectively distributed in the same space according to the need of productions. The linear DHHT assigns a robot R_i to a station S_j , if S_j minimizes the weighted distance $\frac{\|R_i - S_j\|}{w_j}$, where $\|R_i - S_j\|$ denotes the Euclidean distance between the position of S_j and the current position of R_i . With a logarithmic distance measure, the logarithmic DHHT assigns a robot R_i to a station S_j , if S_j minimizes the weighted distance $\frac{\ln(1+\|R_i - S_j\|)}{w_j}$.

If a single station is added or the weight of a station is increased, robots will be reassigned from other stations to these stations. The swapping of robots between other stations is not necessary. Similarly, if a station is removed or the weight of a station is decreased, only this station will free up robots for other stations. So both DHHTs preserve the consistency. The locality is followed by the linear (logarithmic) distance measure used to assign a robot. The shorter the distance between the station and this robot, the lower is the weighted distance of this station.

The situation is much more complicated for keeping the balance which is the most important feature of our approach, because we want to get an even distribution of robots among stations. It is known that this scheme approximates the weighted balance if the stations are randomly distributed in the space [52]. In this case, each station receives a fair share of $\frac{w_i}{\sum_j w_j}$ robots. But this assumption does not hold in our problem. One can try to use multiple copies of slots to solve this problem as mentioned above. However, this does not help in our case because using copies will destroy the locality property.

If we permit the use of placeholder values $w'_1, \dots, w'_n$ for the weights in the DHHT which do not necessarily reflect the desired weighting $w_1, \dots, w_n$, a balanced distribution can be achieved.

Theorem 2.1. *For any placement of stations $S_1, \dots, S_n$ and for all $w_1, \dots, w_n > 0$ there exists a choice $w'_1, \dots, w'_n$ for the linear and logarithmic DHHTs such that the probabilities for a given robot with random placement $p_i(w'_1, \dots, w'_n)$ to be assigned to S_i is $\frac{w_i}{\sum_j w_j}$.*

It is proved that such a choice of weights exists [103] and we will use an iterative, heuristic method to approximate such weights. Additionally, the simulation shows that for non-random locations in two-dimensional environment the logarithmic DHHT performs better. The final version of the DHHT used in this thesis is defined as follows:

The DHHT assigns a robot R_i to a station S_j ,
if S_j minimizes the weighted distance $\frac{\ln(1 + ||R_i - S_j||)}{w_j}$.

2.3.1.1 Station Status Information and Weight

As mentioned above, there exists a choice of weights so that for non-random positions of stations a balanced distribution can be achieved. We use an iterative heuristic method to appropriate such weights. At first, we define a so-called Station Status Information (SSI) to represent the current status of each station, and then, we show, how to compute such a weight for each station.

The SSI consists of a tuple $\langle t, N_Q, N_C, N_{delivered}, e \rangle$, where t is a time stamp indicating the freshness of the information, N_Q the current queue length of tasks, N_C the number of robots currently assigned to tasks in the queue, $N_{delivered}$ the total number of crates delivered, and e a value expressing the station's current delivery efficiency. Note that $N_{delivered}$ has to be bounded by a time window. In our implementation we consider the total number of delivered crates within the last 20 minutes.

We define the *throughput rate* Tr as the number of tasks currently dispatched per time unit by a station. The maximal throughput rate Tr_{max} is given by the minimum between the number of crates arriving at the loading station per time unit (which has to be sampled), and a constant k , where $1/k$ characterizes the dispatching latency, i.e., the time needed by a robot docking with the loading

station, loading the crate, and leaving it. Note that k can vary for each station, for example, due to physical constraints. In our implementation we assume that the k is the same for all stations. The *delivery efficiency* of a station i is then defined by:

$$e_i = \frac{Tr}{Tr_{max}}. \quad (2.1)$$

Therefore, delivery efficiency values close to 1.0 are typical for stations served by robots sufficiently, whereas values close to 0.0 represent stations that have been strongly unattended by robots. We have defined the delivery efficiency to be non-zero and we are using this term as divisor later on. We define $e_i = 1.0$ if $Tr_{max} = 0$, and $e_i = \epsilon$ if $Tr = 0$, where ϵ is a positive constant close to zero.

Each station sends periodically a station status information (SSI) to the network by using beacon messages. Each robot computes the up-to-date weights of all stations with the help of received SSIs. How can such messages be propagated and maintained in the network will be described in Section 2.3.2.

The weight w_i of a station S_i is then computed with respect to the latest SSI received from the station as follows:

$$w_i = \frac{1}{e_i} \frac{N_Q - N_C}{N_{delivered}}. \quad (2.2)$$

In Equation 2.2 the long-term demand $1/e_i$ and the current queue length are main influence factors on the weighting. Furthermore, we shrink the real queue length according to the current amount of robots N_C which are already assigned for the station. Note that stations are immediately aware of the robots due to a so-called request message from robots which declare their interests. The request message will be described in Section 2.3.3. Consequently, they will increase N_C leading automatically to a different weighting and have thus less eligibility compared to other stations.

2.3.2 Mobile Content-Addressable Network

We use an iterative heuristic method to approximate a even distribution of robots among stations guided by the DHHT. In this process, the stations and mobile robots build a decentralized peer-to-peer network in a two-dimensional space. This structure is then similar to Content Addressable Networks (CANs) [86] that provide distributed hash table function to store data in the peer-to-peer network. More precisely, in CANs peers are virtually placed in a two-dimensional square.

Each peer is responsible for a local rectangle of the space and communicates only with its direct neighbors. A data element is assigned to a peer by using a consistent hash function.

Compared to CANs, our problem is defined in the true two-dimensional space, instead of a virtual space. Moreover, mobile robots can change their positions continuously in the space. Thus, we will refer our network specifically to a Mobile Content-Addressable Network (MCAN).

2.3.2.1 Geographic Network Routing

In our setting, we assume that the robots and stations build a mobile peer-to-peer network. The communication link between members in the network can be automatically maintained whenever they are in the direct communication range. The positions of stations are fixed and known in advance, and the positions of mobile robots are also assumed to be known by applying some kind of localization methods. Furthermore, robots exchange their positions with their direct neighbors constantly, whenever the communication link is available. All messages including SSIs, transportation tasks and so on should be delivered between mobile robots and stations without any central infrastructure. But because of the limited communication range and bandwidth, multicast is not feasible. Then, a geographic routing scheme based on the true geographical positions will be used to forward messages in our network [36, 69].

In our work, a simplified variant of the greedy perimeter forwarding routing scheme [53] is implemented. Greedily, each robot forwards a message to its direct neighbor with shortest distance to the destination (see Figures 2.1). If the delivery of a message fails because of a failure of connections caused by moving away of neighbors, the message is then delivered by using a backtracking mechanism as a fall-back strategy. In this case, the backtracking mechanism selects the next shortest path to deliver the message.

Routing efficiency can be improved by adapting the routing metric (Euclidean distance) according to the signal strength. In this case, the robot with higher signal strength will be preferred by the routing scheme, since higher signal strength indicates a reliable connection which will reduce the delivery time of the message. In the CAN the average path length has been estimated by $O(n^{1/2})$ in two-dimensional space for n peers. In our MCAN this might vary, because the robots can be distributed unevenly in the space. In the worst case, the network can be even divided into several parts. But this does not really pose a problem for

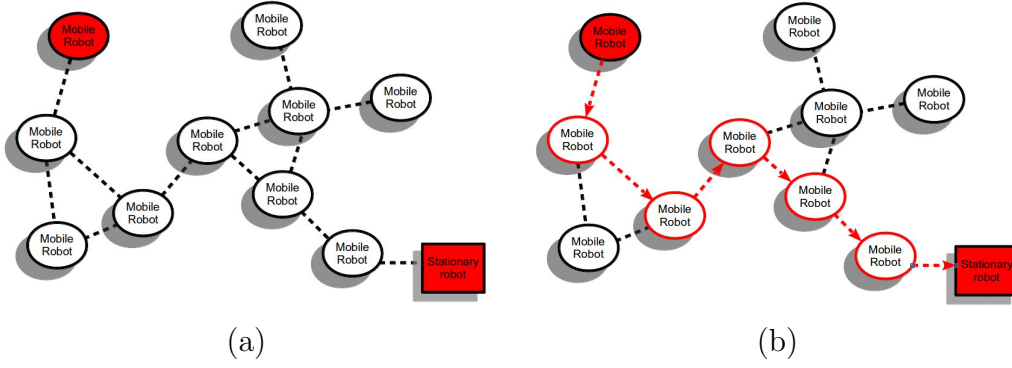


FIGURE 2.1: (a) Black dotted lines denote the direct connections between robots in the MCAN. (b) Routing along the red dotted lines indicate the courses from a mobile robot to a stationary robot (station).

us, because robots can use its mobility to deliver messages and as a side effect of the mobility, the (long-time) receiving range of a robot is much larger than those of static robots.

2.3.2.2 MCAN Construction and Maintenance

In this section, we describe how our MCAN is constructed and maintained. When a robot is added to MCAN, it performs the following three-step procedure to become a member of the network and to be assigned to its first task:

1. **Search for the network.** The robot randomly explores the physical space until it meets the communication range of any other robot connected with the network.
2. **Receive station SSIs.** The robot waits and collects station status messages for a certain time.
3. **Station selection.** Based on the received station SSIs, the robot selects a station as a contracting partner according to the DHHT, which will be explained in Section 2.3.3.

Because of the mobility of robots, the network can be divided into several parts or even become entirely disconnected. The latter situation is particularly the case during bootstrapping, for example, when the first robot enters the environment. Thus, network repairs are necessary in the MCAN. Assuming that the robots know the map of the environment and the list of existing stations in advance, a simple mechanism can be used to repair the network. The time stamp t_i in the

SSI message can be used to measure the age of a message. Note that the age of a SSI is defined as infinite, if it has never been received. If a robot receives an incomplete set of SSIs, or even an empty set, it establishes a connection between the network and a randomly selected station from this set by moving into the station's direction. Consequently, new SSIs will be spread over the network after the connection has been (re-)established. Furthermore, robots cache copies of the messages and use its mobility to deliver messages. This technique is known as delay tolerant routing [47].

2.3.3 DHHT-based Task Allocation

Mobile robots which initially join the network or just finish a previous task, have to be (re)assigned to new tasks. This is carried out by, firstly, deciding a station based on the DHHT, secondly, sending a task request message *REQ* to the selected station, and thirdly, moving towards the station and negotiating for a specific task when within direct communication range with the station. The *REQ* messages are forwarded using geographic routing scheme described in Section 2.3.2.1.

For the first step, the final version of DHHT described in Section 2.3.1 is applied to select a station for a robot, for which the weight of each station at first computed via the Equation 2.2 described in Section 2.3.1.1. Afterwards, the weighted distance for each station with respect to a robot R_i is computed by:

$$D_{ij} = \frac{\ln(d_{ij})}{w_j}, \quad \text{for } d_{ij} > d_{min} \quad (2.3)$$

where w_j denotes the weight of a station S_j , d_{ij} the distance between the robot and station, and d_{min} a minimum radius around the station which should be larger than 1. The d_{ij} can straight-forwardly be computed by the Euclidean distance. However, because the environment can be arbitrarily disconnected with obstacles and walls, this distance metric is an inaccurate estimate. Therefore, d_{ij} is computed by the real travel distance from the robot to the station, which is estimated by a Voronoi Graph [15] computed on the grid map of the environment in advance.

Finally, the station S_j with the shortest weighted distance D_{ij} is selected for the robot R_i and the robot moves to this station. When the robot is arriving in the vicinity of this station, it tries to allocate a new task from this station with a contract net protocol [97]. After the successful allocation of a task, this station decreases N_C and N_Q by one.

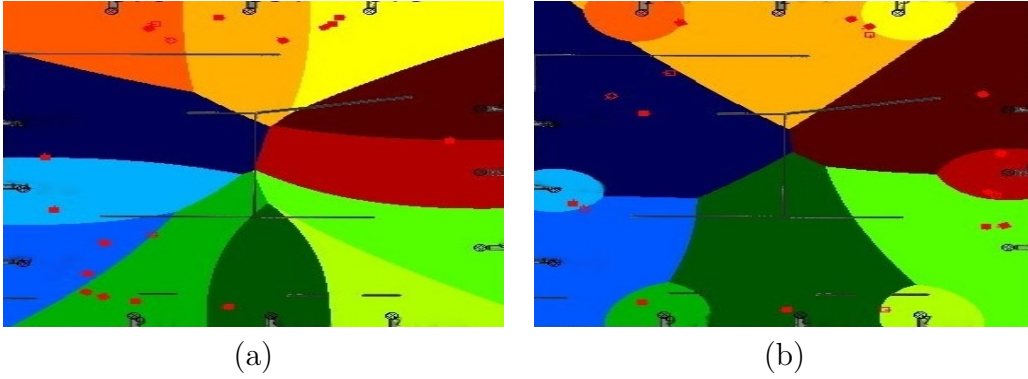


FIGURE 2.2: Visualization of $\min(D_i)$ for two different situations on the *small map*. Weighted distances D_i of each station i are depicted by a specific color. For each location the minimum D_i is drawn.

Figure 2.2 visualizes the effect of Equation 2.3 in two situations. Each station governs a different region for assigning mobile robots. In the specific situation shown in (a) all stations were simulated with a same rate of package arrivals (4 crates per minute), whereas in (b) 6 stations were simulated with a high rate (5 crates per minute), and 6 others with a low rate (2 crates per minute). Consequently, in (a) station areas are evenly distributed, whereas in (b) some stations have more influence than others.

2.4 Experiments and Results

This approach has been evaluated both in simulation and in a real-world scenario. In the simulation the system is simulated by the *USARSim (Unified System for Automation and Robot Simulation)* [14] framework and in the real-world scenario three KARIS robots are used in the experiment. The method is tested according to its convergence behavior towards a fair robot distribution among the stations, and the system stability with regards to rapid changes, such as a change of the rate of package arrivals of each station, and the situation that some robots suddenly fail.

Note in this chapter we do not handle the MRMP problem and assume that this problem is solved by the prioritized multi-robot path planning method [6] discussed in Chapter 4. The focus of this chapter is to evaluate the MRTA approach.

2.4.1 Simulation Results

The proposed approach is evaluated with up to 18 robots in the 3D-simulation simulated by the *USARSim* framework. *USARSim* is an extension of the game engine of the computer game *Unreal Tournament 2004*, and is particularly tailored for simulating heterogeneous robot teams in the context of urban search and rescue (USAR) and automation.

The game engine of UT2004 is utilized to simulate robot navigation and sensor perception in 3D. Several authors validate the simulator with respect to its reality-closeness [4, 46, 89]. In our experiments, the algorithms controlling robots in *USARSim* are also used in real systems. Our experiments are conducted by running in parallel two 8-core Intel Xeon CPUs (2.66GHz), two Dual-core Intel Xeon (3.06GHz), one Dual-core (2GHz), and one Intel Core2Duo (3GHz).

Figure 2.3 (a,c) depict two environments with virtual loading stations used for testing our approach. Figure 2.3 (b,d) depict grid maps generated from these environments with superimposed Voronoi graph utilized for the distance metric. Figure 2.3 (e) depicts a environment created according to the layout of one of the production halls from a larger company, Figure 2.3 (f) depicts the grid map generated from the environment with superimposed Voronoi graph.

The wireless simulation server (WSS) [79] is applied in the experiments, which can simulate signal path attenuation of wireless network transmission between the peers. Therefore, robots which are far from each other or separated by walls experience signal loss, for instance, are impossible to communicate directly with each other.

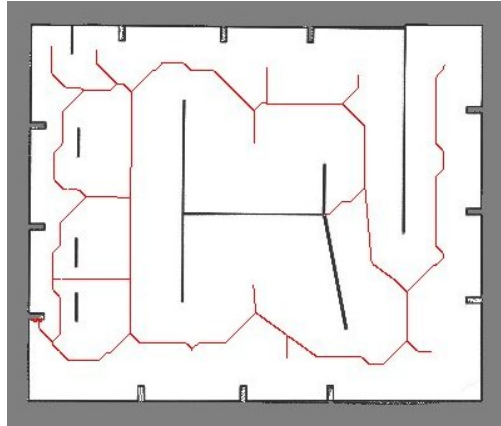
The arrival of crates at loading stations is modeled by a Poisson distribution. Each station is characterized by the rate of package arrivals which is the average number of packages λ_i arriving within one time unit. With each time step of the simulation, the amount k_i of packages generated at station i is stochastically drawn from:

$$P_{\lambda_i}(k_i) = \frac{\lambda_i^{k_i}}{k_i!} e^{-\lambda_i}, \quad (2.4)$$

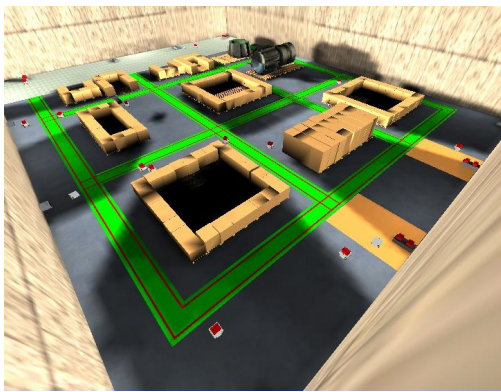
Likewise to an assembly line in the real setting, arriving crates are kept in a virtual queue attached to each station. Note that in our experiments every station is initiated with a number of tasks in advance. Therefore, the virtual queue is not empty at the beginning of the experiments. Each time the oldest crate from



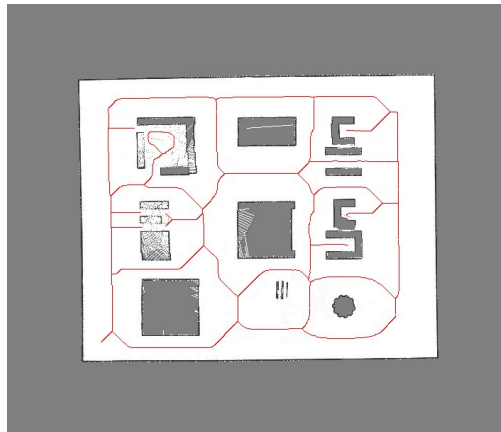
(a)



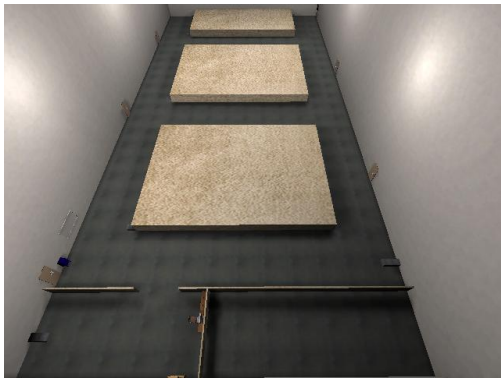
(b)



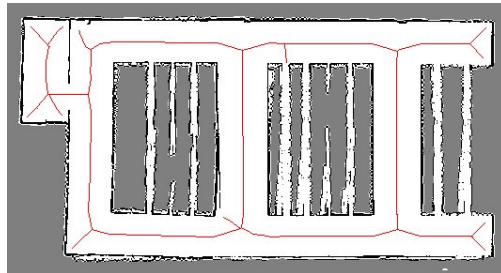
(c)



(d)



(e)



(f)

FIGURE 2.3: (a,c) Simulated environments in USARSim. (d,e) Generated grid maps for localization and planning with superimposed Voronoi graph (red). Maps are denoted as (a) *Small factory*, (c) *Big factory* and (e) *Company map*.

the queue is automatically loaded by a robot which has already been docked with the station, and will be delivered to its destination by this robot. The destinations of crates are drawn stochastically from a Gaussian distribution with mean $\mu = (s_x, s_y)$ according to the station's location, which means that these stations are close to the source will be chosen with higher probability than others. This

TABLE 2.1: Simulation Results DHHT & Baseline

	Small factory	Big factory	Company map
Base Eff. [%]	0.64 ± 0.27	0.41 ± 0.27	0.8 ± 0.11
DHHT Eff. [%]	1.0 ± 0.02	0.64 ± 0.03	1.0 ± 0.0
Base Deliv. [#]	115.2 ± 44.0	77.4 ± 46.0	155 ± 22.2
DHHT Deliv. [#]	178.6 ± 1.6	151.4 ± 9.6	177.1 ± 1.2
Base WCTD [min.]	71.6	50.1	75.6
DHHT WCTD [min.]	3.3	16.2	2.4

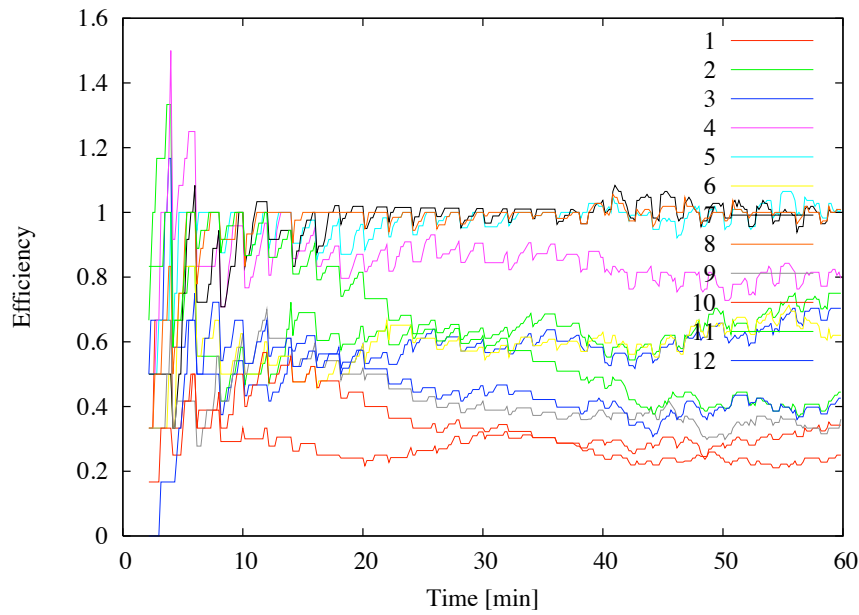
simulates the situation in a real plant where correlated products are located more likely close to each other.

2.4.1.1 Efficiency and Convergence

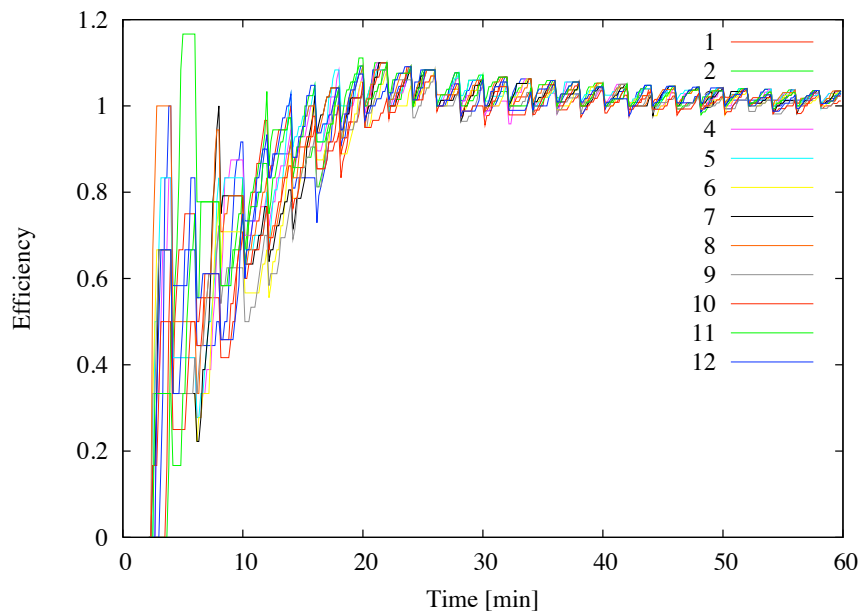
We compare the proposed DHHT-based task allocation method with the straightforward solution denoted by *Baseline*. The baseline is to assign robots, according to their spacial distance, to stations. Robots receive task offers from all stations and make a decision for the station with the shortest distance. Note that in general, the baseline method does not automatically lead to the situation where robots only work for a subset of stations. Because of the Gaussian distribution of the destinations of crates, after some time some robots might reach any station when accomplishing delivery tasks with far destinations. However, it is not guaranteed that they assign themselves evenly to the stations. Some stations can still be disadvantageous due to their location.

Table 2.1 shows the average delivery efficiency over all stations (Eff.), the total number delivered packages (Deliv.) and the worst-case time-delay of the delivery (WCTD). The delivery efficiency of a single station is calculated with the equation 2.1. As we can see from the results, the DHHT-based method reaches a higher average delivery efficiency (DHHT Eff. 1.0, 0.64, and 1.0), , whereas the average delivery efficiency of the baseline (Base Eff.) is only 0.64, 0.41, and 0.8 respectively. The total number of delivered packages (Deliv.) of our method are also better in comparison to the baseline. Furthermore, the lower standard deviation of Eff. and the better WCTD of our method indicate that the robots are more evenly distributed among the stations then that in baseline. This can be clearly seen from the figures 2.4 (Small map) and 2.5 (Company map), which show the delivery efficiency of each station over time. In our method (figures (b)), stations are receiving a fair share of robots during the running, which means that our method has a better degree of fairness. Note that sometimes the delivery

efficiency is bigger than 1.0, due to the fact that the virtual queue of stations is initiated with a number of tasks.



(a)



(b)

FIGURE 2.4: Efficiency of all 12 stations over time on the *Small map*. Each color denotes a single station. (a) Baseline approach, and (b) DHHT approach.

2.4.1.2 Adaption to sudden changes

Within another series of experiments we evaluate the behavior of our approach in case of sudden changes, for example, the rate of package arrivals changes or some

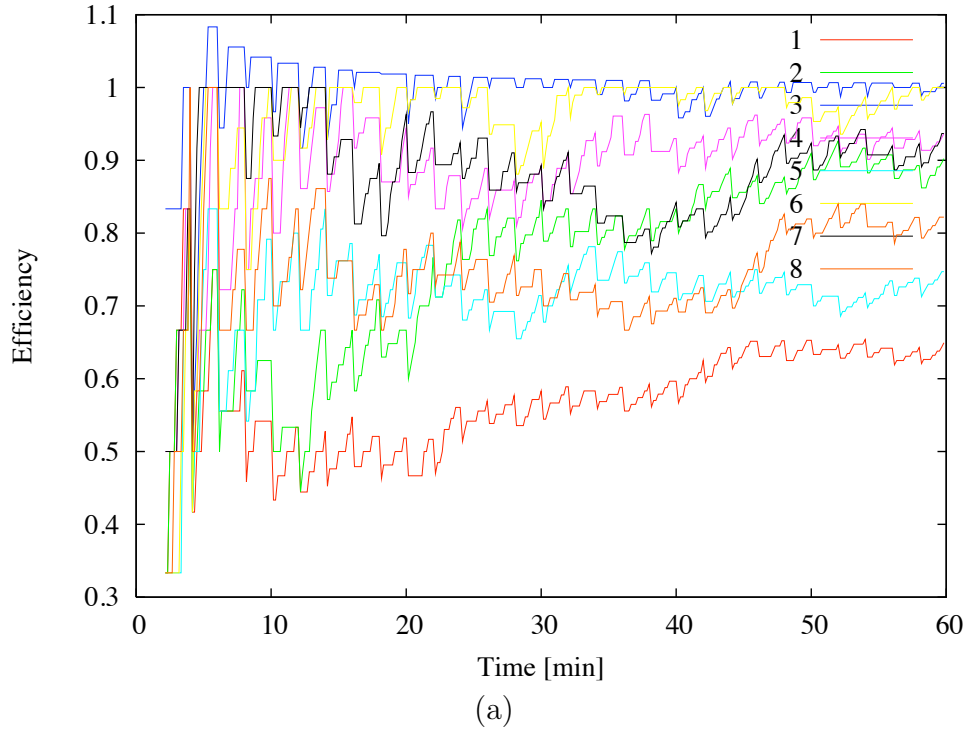
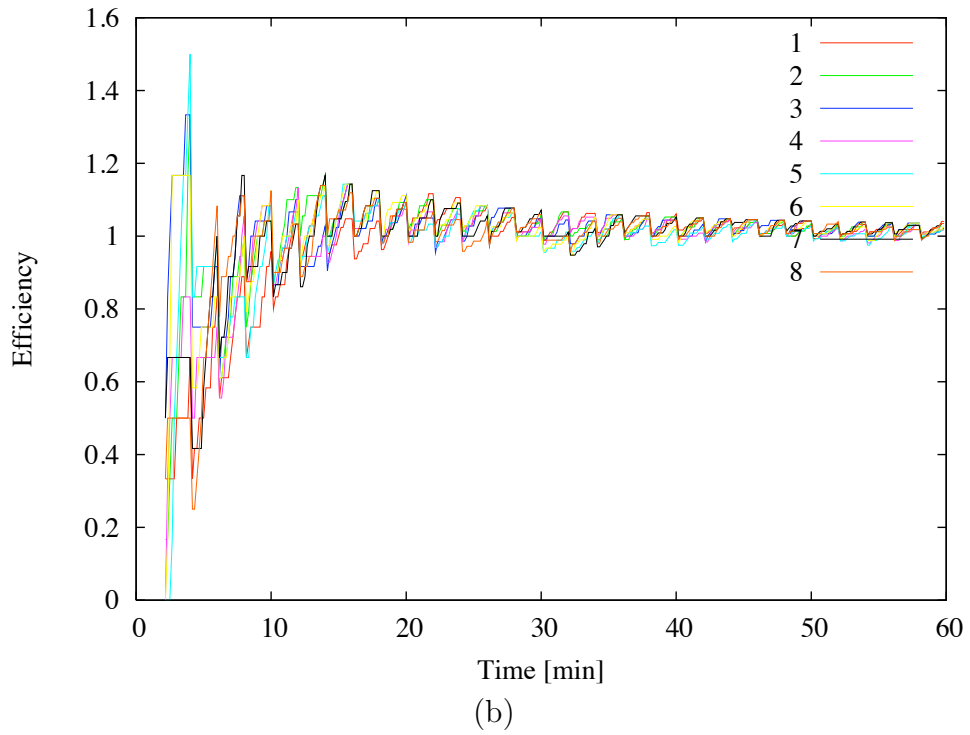


FIGURE 2.5: Efficiency of all 8 stations over time on the *Company map*. Each color denotes a single station. (a) Baseline approach, and (b) DHHT approach.

robots crash. Note that all experiments are repeated for several times. Figure 2.6 depicts the delivery efficiency of stations over time. After 30 minutes the rate of package arrivals at each station is suddenly changed. Instead of $\lambda = 8$ we change

for 6 stations λ to 10 and for the other 6 stations λ to 4. The result show that after the change, the delivery efficiency of stations with lower rate increases and the one with higher rate decreases. However, after some time all stations converge to an evenly distributed delivery efficiency as enforced by our method.

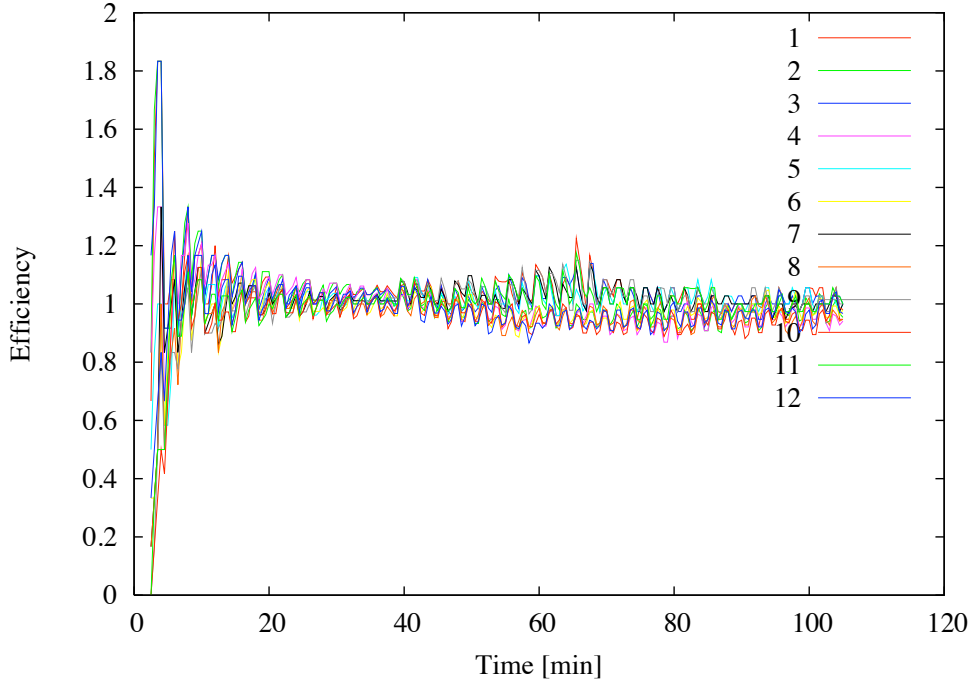


FIGURE 2.6: Adaption after rapid throughput change. For the first 30 minutes all 12 station were running at $\lambda = 8$, and from then on 6 with with $\lambda = 10$ and 6 with $\lambda = 4$

Figure 2.7 depicts the result from an experiment where the number of robots has been suddenly decreased. We run 18 robots on the *small map* and simulate a robot crash by removing 10 of them after 30 minutes. The figure suggests that the team did react on the change, namely all stations are continuously visited by robots. Obviously, the overall delivery efficiency (mean) and the overall fairness (standard deviation) decrease slightly due to the change. The fairness has to decrease since the reduced set of robots is forced to cover longer distances on the map in order to serve all stations.

2.4.2 Real-World Results

Finally, we conduct real-world experiments with three KARIS [42] robots (see Figure 2.8). The second prototype of the KARIS robot has a size of $50 \times 50 \times 45$ (length x width x height) cm with a payload of 60 kg. The robot is equipped with

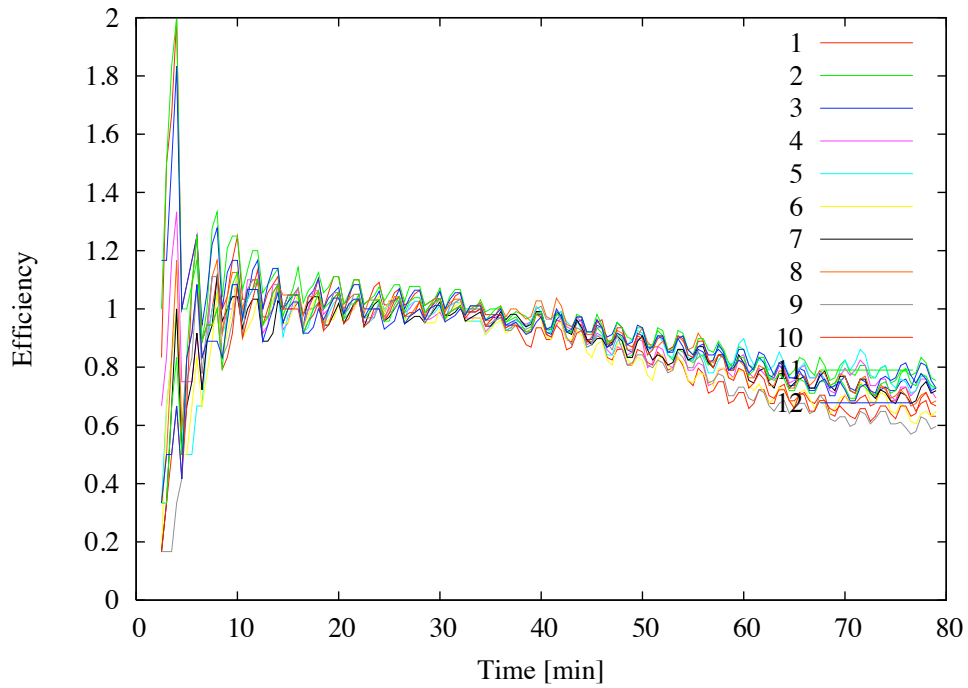


FIGURE 2.7: Adaption after a sudden break-down. For the first 30 minutes the system was running with 18 robots, then, we intentionally crashed 10 of them.

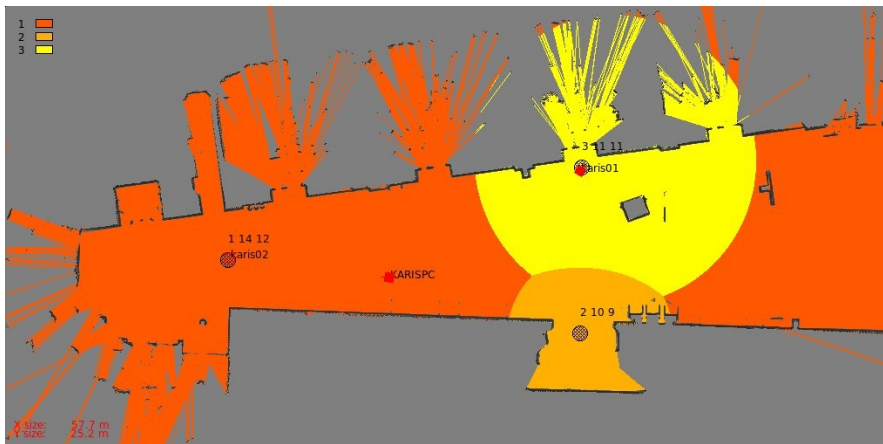
a holonomic drive to facilitate docking behaviors. For the purpose of autonomous navigation, the robot is equipped with two SICK S300 laser range finders (LRFs) mounted in two opposing corners, wheel odometry, and an inertial measurement unit (IMU). The navigation system used in this experiment is based on A* [88] planning on occupancy grid maps which is generated from the data collected by once steering a single robot manually through the environment. We use Monte-Carlo localization [18] with wheel odometry, IMU, and range readings from the two LRFs to localize the robot on the occupancy grid map.

Stations are simulated in the same way as described in Section 2.4.1. In all experiments the robots run autonomously more than five hours in total without causing collisions or dead-locks. They finish the transportation tasks for up to four stations, with one station placed relatively far from the others. Figure 2.9 depicts the map of the testing environment with stations marked as circles. The coloring shows the weighted distance computed for each station. In the particular moment indicated in the figure, station 1, located on the far left end, has the strongest influence (dark orange) on the robots due to the fact that DHHT balance its outer location.

Table 2.2 summarizes the results from two different settings. As can be seen, the DHHT assignment reaches a fairer distribution (standard deviation) and a smaller



FIGURE 2.8: The second prototype of KARIS robot.

FIGURE 2.9: Map of the real-world scenario and visualization of $\min(D_i)$. The far-off station on the left side (dark orange) has the highest demand for robots.

worst-case time-delay of deliveries.

Figure 2.10 depicts the delivery efficiency over time, which is also more evenly distributed here when the DHHT-based method is applied. In real-world experiments, it is also noticed that the system reacts appropriately to changes, i.e., it redistributes its resources. For example, in one situation, a robot runs out of power, and in another we increase the travel time to one station by blocking the path.

TABLE 2.2: Real-World Results DHHT & Baseline

	Three stations	Four stations
Base Eff. [%]	0.69 ± 0.1	0.74 ± 0.24
DHHT Eff. [%]	0.76 ± 0.05	0.9 ± 0.09
Base Deliv. [#]	36.3 ± 4.9	20.3 ± 6.5
DHHT Deliv. [#]	24 ± 1.6	25.5 ± 2.7
Base WCTD [min.]	20.1	18.3
DHHT WCTD [min.]	14.5	10.5

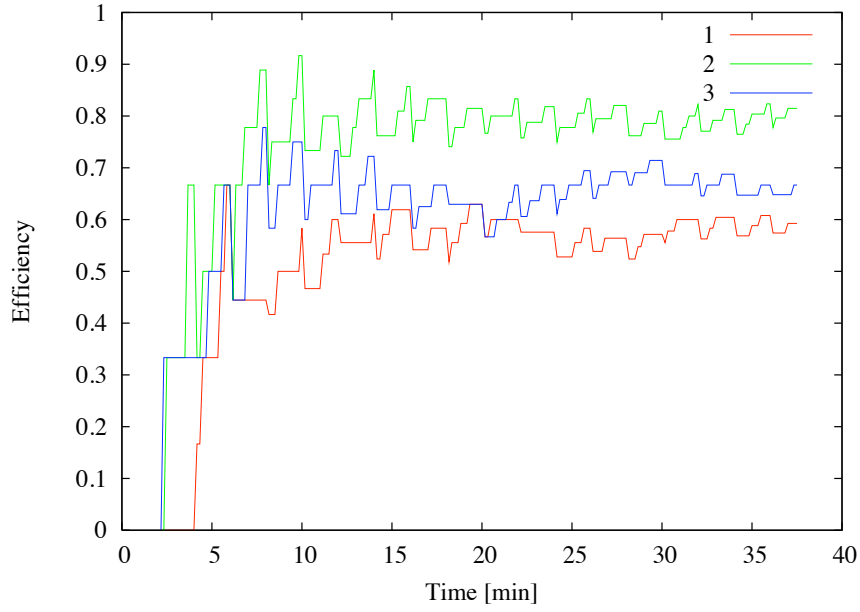
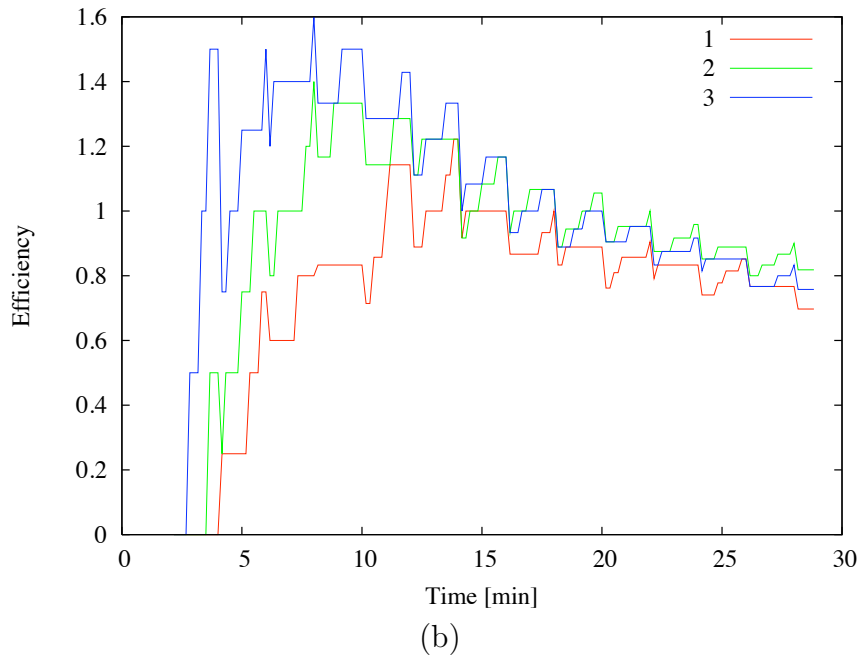


FIGURE 2.10: Efficiency of all 3 stations over time in the real environment. (a) Baseline approach, and (b) DHHT approach.

2.5 Conclusion

In this chapter, we introduce a novel approach, called DHHT, to solve the tasks allocation problem in intra-logistics domains. The goal is to assign robots to a set of transportation tasks offered by stations. The assignment of a robot to a task is divided into two steps in our approach: firstly, a robot is assigned to a station by using the DHHT method. Then, the robot gets a specific transportation task from this station with the help of a contract net protocol. The robots build a Mobile Content-Addressable Network (MCAN) and in the MCAN all messages are delivered via a so-called geographic routing scheme.

Our approach is completely decentralized, which means that it needs no central server to coordinate the communication or the task allocation. Only robots and loading stations carry communication devices and all necessary communications are performed by the robots and the loading stations. Therefore, our approach is applicable in many production environments with no necessity of installing additional hardware infrastructures. Moreover, due to the decentralized structure of the algorithm, the system is robust, fail-tolerant and scalable. Failures of some robots or adding further robots during the runtime has only marginal influence on the system, and the changing demands of stations can be easily adapted by the system. Besides the theoretical findings for the allocation mechanism, we evaluate our approach in both simulation and real world. The experimental results from the simulation and real world have shown that our approach reaches a fair distribution of robots among stations under several different constraints, and thus achieves the higher delivery efficiency of the transport system.

Chapter 3

Localization in Dynamic Environments

3.1 Introduction

Localization is one fundamental problem in mobile robotics. Autonomous navigation requires that at first a mobile robot can determine its position in an environment. In outdoor scenarios, one can use an accurate global position system (GPS) sensor to solve the localization problem. But the existing GPS network provides only limited accuracy which is not good enough for some small-scale robots or applications. Furthermore, GPS technologies cannot function indoors. So the more common way is to use sensors to perceive the environment and to build a map, then to identify a robot's position relative to that map. There are a wide variety of sensors used to acquire information about the environment, for example, tactile sensors to detect physical contact, active ranging sensors to measure distance to nearby objects, vision-based sensors to analyze light intensity or color values [94, Chapter 4]. In consideration of the noise level and the cost, a 2D laser range finder is used in our work to perceive distance information in indoor environments.

Depending on what is known about the environment at the beginning, the localization problem can roughly be divided into two main types: localization in a known environment and localization in an unknown environment. Localization in an unknown environment is commonly referred to as Simultaneous Localization and Mapping (SLAM) [39, 72] problem. In this thesis, we assume that the environment is known and concentrate on the problem of keeping track of a robot's location within the map. There are several ways to represent the environment.

Map representation can usually be divided into two classes: continuous representations and decomposition strategies. Choosing a particular map representation is normally dual of the problem of the representation of the robot's configuration space and the sensors. For the robot equipped with range-based sensors, the most common map representation is the fixed decomposition strategy. One very popular version of this strategy is known as the occupancy grid map [73]. In this strategy, the environment is represented as a discrete grid, where each cell is of the same size and is either occupied by a part of an object or free. According to the requirements of accuracy, the size of the cell can be enlarged or reduced. Therefore, the size of the map is dependent not only on the size of the environment but also on the fine degree of cells, and it can be very large compared to continuous map representations. In spite of that, the range sensor based localization can benefit from this method, because the range values from the sensor combined with the robot pose can be used directly to update the occupancy state of each cell. The work throughout this thesis is based on the occupancy grid map representation. For more details about the map presentation, please refer to [94, Chapter 5].

According to the knowledge of the initial robot pose, localization within a known environment can be classified into three categories [107, Chapter 7]. The simplest is the pose tracking problem, which assumes that the initial robot pose is given. Localization seeks to correct small, but incremental odometry errors. The uncertainty is local and restricted to a small region near the robot's true pose. What is more difficult is the global localization, where the initial robot pose is unknown. The robot has to at first determine the initial pose and then keep track of its pose. Since the determination of the initial pose cannot be guaranteed to be free from error, the localization error can be arbitrarily large. What is even more challenging is the kidnapped robot problem, in which an already localized robot is teleported to somewhere else without being aware of it. This problem is more difficult than global localization, because the robot is firmly convinced of a wrong belief of its pose.

In static environments, many approaches have already been implemented to solve the pose tracking and global localization problem successfully, for example, the Monte Carlo localization [107, Chapter 8]. By adding random particles in the sampling process, a so-called augmented Monte Carlo localization can also solve the kidnapped robot problem well. But real environments are rarely static, especially some environments like intralogistic centers or production halls, are even highly dynamic. Changes in such an environment occur continuously over time and can be very huge after a while. The localization methods which do not take dynamic influences into account can often fail in extreme cases.

In this chapter, we consider the pose tracking problem in highly dynamic environments and introduce an effective and robust dynamic localization method. Similar to SLAM, our method separates localization from the mapping process and updates map when changes in the environment are detected. But it is worth noting that our problem is different from the SLAM problem from two aspects: a map of the environment is known at the beginning, and the environment is dynamic. In the localization process, we apply a Monte Carlo localization method coupled with a distance filter and a scan matching method to achieve a more robust localization to overcome dynamic obstacles. In the mapping process, we take dynamic influences and the state transition probability into account to extend a normal sensor model [71], which leads to a more accurate mapping result. Combining localization with map updating, our approach achieves a more accurate and robust localization in highly dynamic real environments.

The remainder of this chapter is organized as follows: in Section 3.2 we present related work. Section 3.3 formally defines our problem of dynamic localization and gives necessary background for Monte Carlo localization with the scan matching and distance filter. Section 3.4 describes the dynamic mapping process and the extended sensor model in detail. Finally, we evaluate our method for multiple real world environments in Section 3.5.

3.2 Related Work

A common approach to deal with dynamic obstacles that only occur for short periods of time is to treat them as outliers. Fox et al. [30] apply, among other things, a *distance filter* to remove readings of a scan which are shorter than the expected value and are therefore caused by an object that is not modeled. Inspired by this work, we not only filter outliers which are shorter than expected, but also detect those that are longer than expected during the localization. And additionally, we use the information of outliers to update the map accordingly.

When changes take place irregularly and the map is being changed for a longer period of time, a distance filter is not sufficient. Tipaldi et al. [108] show that even if their SLAM algorithm is applied permanently and loop closure techniques join different trajectories together, the generated map gets highly inconsistent and the trajectory path is wrong. Thus, they treat the lifelong localization problem as a multi-session localization and mapping problem instead. Their map is based on a *dynamic occupancy grid* [71], which consists of a collection of individual cells modeled with a hidden Markov model. The grid requires a state transition probability,

corresponding to the (static) probability that a cell changes its state. Tipaldi et al. separate the estimation of the robot trajectory from the map estimation and use a particle filter to compute the robot trajectory, where each particle is associated with its own map. They reduce the size of these local maps by exploiting properties of the hidden Markov model. However, since the number of the particles required for global localization amounts to several thousand, memory management still plays a key role.

Another way to achieve global localization is to use scan matching instead of a particle filter. Recently, Olson [76] present a multi-resolution scan matching method, which generates the same result as brute-force full resolution methods, but runs an order of magnitude faster than previous approaches. This technique is well-suited when the initial map of the environment is known.

Our approach also separates mapping from localization. We use a particle filter localization method and apply scan matching afterwards, to improve the accuracy of the estimated pose. However, we don't associate a map to each particle. Instead, we initiate a dynamic mapping process, to update the map if the uncertainty of the estimated robot pose is low. This prevents map updates which are based on wrong estimates. The updated map will then be used for upcoming the localization process.

Other work on localization in dynamic environments represents dynamic objects as explicit object models on the map. Anguelov et al. [3] extract dynamic object snapshots in office-type environments from static occupancy grid maps acquired at different times. They use an EM-algorithm [70] to learn two-level hierarchical shape models, which links these object snapshots to generic shape templates of several object classes. Similarly, Biswas et al. [8] also apply static occupancy grid maps to extract object snapshots and use an EM-algorithm to learn object models represented as local occupancy grid maps. Both of these methods have achieved significant accuracy at modeling dynamic objects in environments. However, they suffer from disadvantages of the off-line algorithms and the limited complexity of objects.

Wang et al. [117] present two approaches to manage localization, mapping and moving object tracking (SLAMMOT) simultaneously. In their first approach, all objects (including robot pose) are modeled with a hybrid state model consisting of a state and a motion mode. They compute a joint posterior over all objects, similar to other SLAM methods [72]. But due to the required motion modeling of all objects, this approach is computationally more complex and thus generally does not suit real-time applications. In their second approach the estimation

problem is decomposed into two separate estimators, namely, for stationary objects and moving objects. They demonstrate that this satisfies navigation and safety requirements in high speed autonomous driving applications in urban areas.

Gallagher et al. [32] present an online approach to deal with dynamic changes. Their map consists of a static occupancy grid map and several lists of dynamic objects. Dynamic objects will be detected and classified via laser observations and recognized objects are saved in lists with positions and orientations. These lists will be updated whenever the map is changed. They have shown that their resulting map improves both localization and navigation.

In contrast to these explicit modeling approaches we integrate changes caused by dynamic objects directly into the occupancy grid mapping, which leads to the independence of our approach from the shape, type and complexity of dynamic objects, and therefore attains its independence from the different type of environments as well.

3.3 Dynamic Localization

The problem of dynamic localization is to calculate the posterior distribution of map configurations $m_{1:t}$ and robot poses $x_{1:t}$ up to time step t , given observations $z_{1:t}$, odometries $u_{1:t}$, initial map configuration m_0 and initial robot pose x_0 , i.e. to estimate $p(m_{1:t}, x_{1:t} \mid z_{1:t}, u_{1:t}, m_0, x_0)$. The major difference from the SLAM problem is that m_0 and x_0 are prior distributions known in advance and that the map configuration changes over time, since the environment is dynamic. Similar to SLAM, we can separate the estimation of the robot pose from the map configuration, by using Bayes' rule and the Markov assumption:

$$\begin{aligned} p(m_{1:t}, x_{1:t} \mid z_{1:t}, u_{1:t}, m_0, x_0) = \\ p(m_{1:t} \mid x_{1:t}, z_{1:t}, m_0, x_0) p(x_{1:t} \mid z_{1:t}, u_{1:t}, m_0, x_0). \end{aligned} \quad (3.1)$$

Figure 3.1 describes the Bayesian network that models the factorization.

Afterwards, we have two separated problems, where $p(m_{1:t} \mid x_{1:t}, z_{1:t}, m_0, x_0)$ corresponds to the problem of mapping with known poses and $p(x_{1:t} \mid z_{1:t}, u_{1:t}, m_0, x_0)$ corresponds to the localization problem. We will discuss the mapping problem in Section 3.4 with prior concentration on localization.

We use a particle filter approach to solve the basic localization problem, and apply a scan matching method to improve the accuracy of the estimated position. For

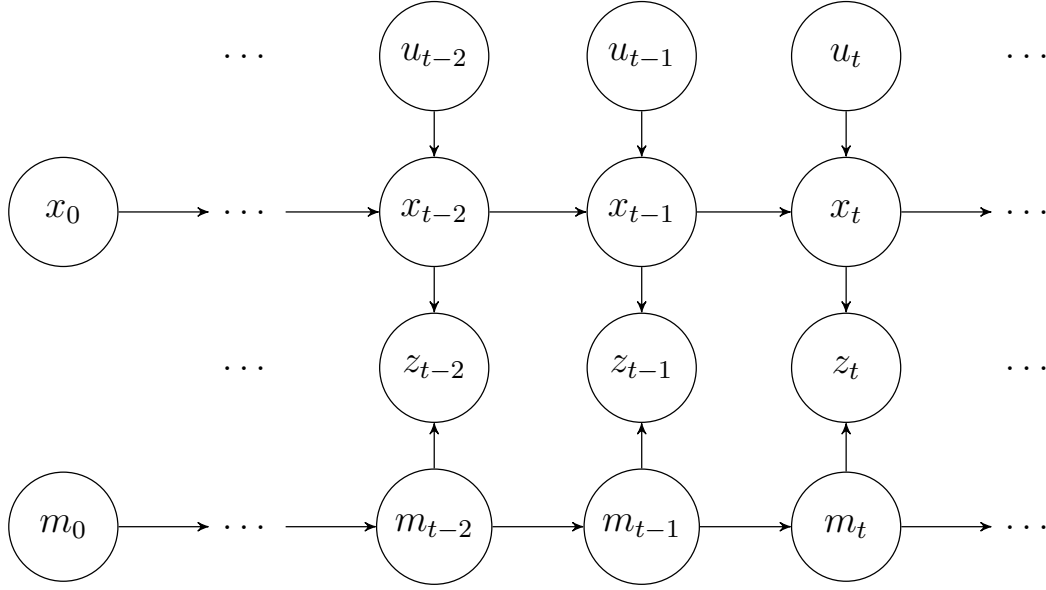


FIGURE 3.1: Bayesian network of the dynamic localization problem.

scan matching, we employ the multi-layer searching technique by Olson [76] to find the local maximum of the position within a limited region. Since this method is extremely fast for small regions, we can substantially increase the accuracy of the localization with requiring only a few computational resources. Additionally, a matching score returned by the this process can be used as a parameter to prevent updates of the map based on wrong estimates. Only if the score exceeds a specific threshold, the map update step will be evoked. For our experiments, this threshold is determined empirically. We will discuss other possibilities to determine it in Section 3.6.

3.3.1 Distance Filter

To account for dynamic obstacles, a distance filter marks observations that are shorter or longer than the expected observation as outliers. Originally, the distance filter by Fox et al. is used to filter those readings of the laser scan that are shorter than the distance expected from the map. Let $d_1, \dots, d_n$ be a set of possible distances and $p_m(d_i|x_l)$ the probability of measuring distance d_i , if the robot is at position x_l . The probability that distance d_i is shorter than expected is defined by

$$p_{short}(d_i|x_l) = \sum_{d_j > d_i} p_m(d_j|x_l),$$

based on the fact that this is equivalent to the probability that the expected result of measurement is longer than d_i . Since the position x_l is based on the current

belief, we have to average over all possible positions of the robot:

$$p_{short}(d_i) = \sum_l p_{short}(d_i|x_l) \cdot p(x_l),$$

where $p(x_l)$ is given by our particle filter. Given $p_{short}(d_i)$, we can now mark those measurements d_i as shorter outliers, where $p_{short}(d_i)$ exceeds some threshold γ . And the shorter outliers will be rejected during localization.

Similar to shorter outliers, the probability that distance d_i is longer than expected is defined by

$$p_{long}(d_i|x_l) = \sum_{d_j < d_i} p_m(d_j|x_l),$$

and the average over all possible position we have is:

$$p_{long}(d_i) = \sum_l p_{long}(d_i|x_l) \cdot p(x_l).$$

We can then mark those measurements d_i as longer outliers if $p_{longer}(d_i)$ exceeds some threshold γ . The longer outliers will not be handled during the localization. But in the dynamic mapping process, both shorter and longer outliers will be applied to influence the rate of the map updating. The details will be described in Section 3.4.

We should mention here that Fox et al. precompute the distance to the closest obstacle on the map for each possible robot location. However, since our environment is not static, we have to compute this distance on the fly. While this is the main bottleneck of our distance filter, our approach still works well in practice.

3.4 Dynamic Mapping

The second problem to tackle is to compute the full posterior distribution of $p(m_{1:t} | x_{1:t}, z_{1:t}, m_0, x_0)$. This computation is not only intractable, but also unnecessary in practice, because we are only interested in the last configuration of the map. Therefore, we use a Bayesian filter to recursively compute the marginal distribution $p(m_t | x_{0:t}, z_{1:t})$ for the current map m_t .

A common approach is to decompose the problem into several one-dimensional estimation problems [74, 106], with the underlying assumption that grid cells are conditionally independent from each another. Therefore, we approximate $p(m_t |$

$x_{0:t}, z_{1:t})$ with $\prod_{i=0}^N P(c_t^i | z_{1:t}^i)$, where $z^i \in \{hit, miss\}$, $c \in \{occ, free\}$ and $i = 1, \dots, N$, N being the total number of cells.

To represent dynamic changes, we use dynamic occupancy grid maps introduced by Meyer-Delius et al. [71], which apply the theory of hidden Markov models (HMM) [85]. In the original work, the state transition model $p(c_t^i | c_{t-1}^i)$ is assumed to be stationary (i.e., the state transition probabilities do not change over time) and can be learned offline or online. We adopt the same assumption and learn it offline. We also adopt the sensor model introduced along with dynamic occupancy grids, which takes two cases into consideration: a measurement ends up in a cell (*hit*) or goes through it (*miss*). However, this sensor model does not consider dynamics influences in the environment. The probability for a measurement *miss*, given that a cell is occupied (i.e. cell state *occ*), is usually quite low, due to the high accuracy of today's laser range finders. But this probability should increase when we know that some objects may disappear in the environment. Similarly, the probability for a measurement *hit*, given that a cell is free, is usually very low. But it should increase when we know that some objects may appear in the environment.

To achieve this, we incorporate information about the transition dynamics of each cell: st^i is a boolean, time-independent flag that specifies whether cell c^i is a static cell, or whether objects may appear or disappear. For example, in a production hall we might flag walls and machinery equipment as static, while hallways, movable shelves and stock in a storeroom are flagged as dynamic. We may learn such information offline, but usually we can also manually flag the map by studying a layout plan of the facility. The second information we incorporate are the results from the distance filter. We consider not only whether a measurement hits a cell c^i , but also if this measurement is regarded as an outlier: $\hat{z}_t^i \in \{hit, miss, hit_o, miss_o\}$. Figure 3.2 represents the underlying HMM of our model.

As a result, we have to compute $\prod_{i=0}^N P(c_t^i | \hat{z}_{1:t}^i, st^i)$. By applying the Bayesian rule we get

$$P(c_t^i | \hat{z}_{1:t}^i, st^i) = \frac{P(c_t^i, \hat{z}_{1:t}^i, st^i)}{P(\hat{z}_{1:t}^i, st^i)},$$

where $P(\hat{z}_{1:t}^i, st^i)$ is independent from the state of the cell. Therefore, we can substitute it by a normalization constant μ , which leads to

$$P(c_t^i | \hat{z}_{1:t}^i, st^i) = \mu P(c_t^i, \hat{z}_t^i, st^i).$$

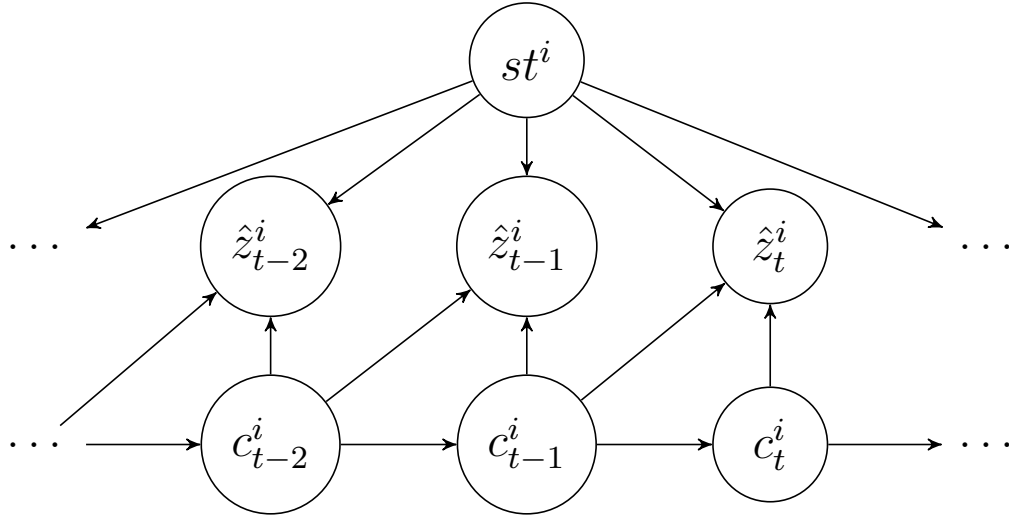


FIGURE 3.2: Hidden Markov Model of the extended sensor model.

According to the Hidden Markov Model 3.2, the state of cell c_t^i also depends on the hidden variable c_{t-1}^i . So by applying the forward algorithm, we get

$$P(c_t^i | \hat{z}_{1:t}^i, st^i) = \mu \sum_{c_{t-1}^i} \cdot P(\hat{z}_t^i, c_t^i, c_{t-1}^i, \hat{z}_{1:t-1}^i, st^i).$$

Applying the chain rule results in

$$\begin{aligned} P(c_t^i | \hat{z}_{1:t}^i, st^i) &= \mu \sum_{c_{t-1}^i} P(\hat{z}_t^i | c_t^i, c_{t-1}^i, \hat{z}_{1:t-1}^i, st^i) \\ &\quad \cdot P(c_t^i | c_{t-1}^i, \hat{z}_{1:t-1}^i, st^i) \\ &\quad \cdot P(c_{t-1}^i | \hat{z}_{1:t-1}^i, st^i) \cdot P(\hat{z}_{1:t-1}^i, st^i). \end{aligned}$$

Due to conditional independence

$$\begin{aligned} P(\hat{z}_t^i | c_t^i, c_{t-1}^i, \hat{z}_{1:t-1}^i, st^i) &= P(\hat{z}_t^i | c_t^i, c_{t-1}^i, st^i), \\ P(c_t^i | c_{t-1}^i, \hat{z}_{1:t-1}^i, st^i) &= P(c_t^i | c_{t-1}^i), \end{aligned}$$

we get

$$\begin{aligned}
P(c_t^i | \hat{z}_{1:t}^i, st^i) &= \mu \sum_{c_{t-1}} P(\hat{z}_t^i | c_t^i, c_{t-1}^i, st^i) \\
&\quad \cdot P(c_t^i | c_{t-1}^i) \\
&\quad \cdot P(c_{t-1}^i | \hat{z}_{1:t-1}^i, st^i) \\
&\quad \cdot P(\hat{z}_{1:t-1}^i, st^i).
\end{aligned}$$

Furthermore, $P(\hat{z}_{1:t-1}^i, st^i)$ is independent from the state of the cell, so it can be substituted by another normalization constant η , which leads to

$$P(c_t^i | \hat{z}_{1:t}^i, st^i) = \mu \eta \sum_{c_{t-1}} P(\hat{z}_t^i | c_t^i, c_{t-1}^i, st^i) P(c_t^i | c_{t-1}^i) \cdot P(c_{t-1}^i | \hat{z}_{1:t-1}^i, st^i).$$

This equation can be computed recursively: $P(\hat{z}_t^i | c_t^i, c_{t-1}^i, st^i)$ corresponds to the sensor model, $P(c_t^i | c_{t-1}^i)$ corresponds to the state transition model and $P(c_{t-1}^i | \hat{z}_{1:t-1}^i, st^i)$ is the recursive call, where the base case is given by the initial map m_0 . Both the sensor model and the state transition model can be learned offline beforehand, as it is done in our experiments (cf. Table 3.1). It should be noted that parameters of the extended sensor model influence the rate at which a cell may switch its state. If the learned parameters are therefore unreasonable, they may lead to inconsistency of the map.

Another advantage of the extended sensor model is that dynamics are not modeled as explicit object models [3, 8, 39, 117]. Our model is independent from the shapes and types of the dynamic objects. This allows us to apply the sensor model learned for one environment to another environment, as we will see in our experimental results.

3.5 Experimental Results

To evaluate our approach, we conducted a series of experiments in two real production halls (PH1: 104m x 52m, PH2: 47m x 26m) and an intralogistic center (LC: 137m x 99m) from three different companies. Changes in the environments may be caused, for example, by forklifts or people moving storage boxes. We are also confronted with more long-term structural changes, for example, the rearrangement of shelves. In each environment, we collected several data sets with a mobile robot equipped with a SICK S300 laser range finder. Each data set consists

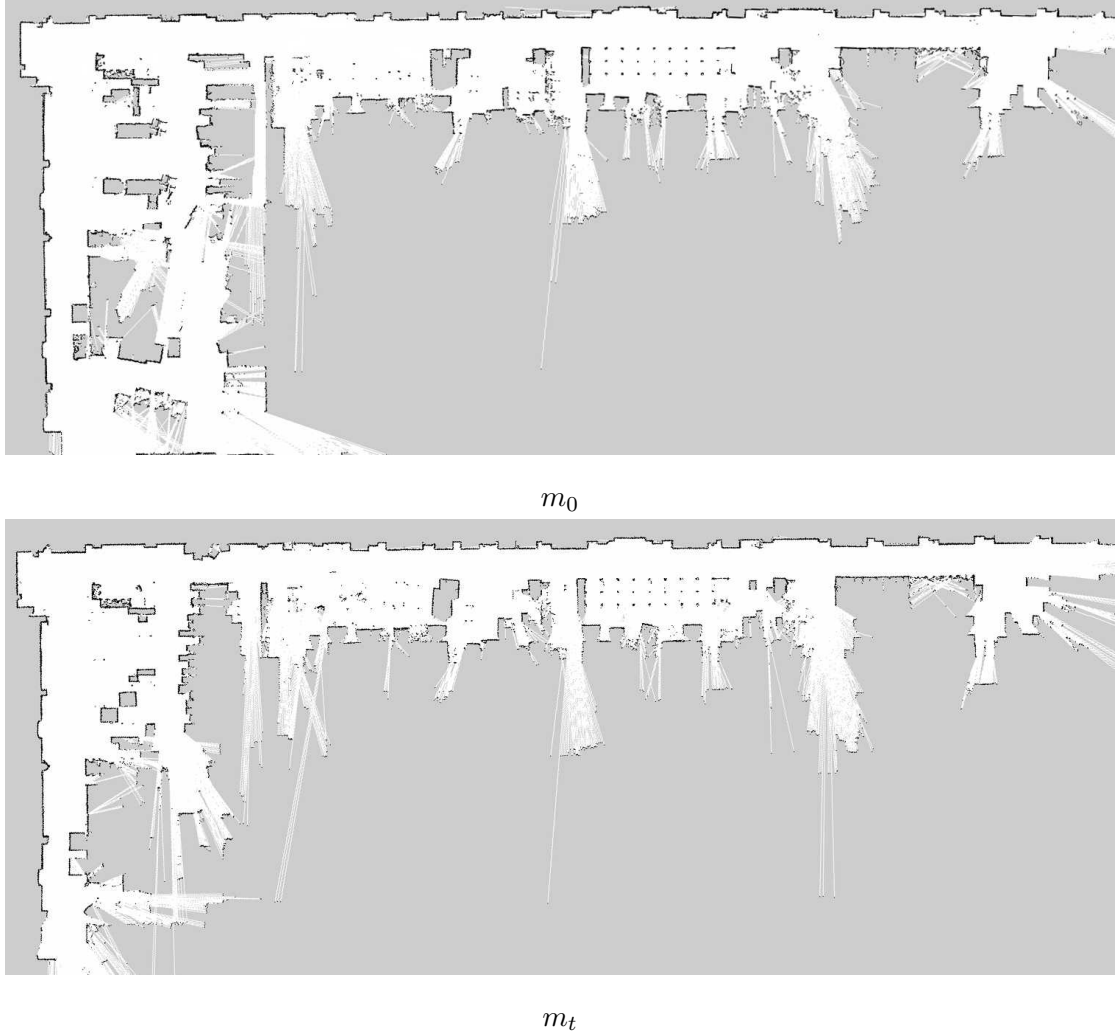


FIGURE 3.3: Ground truth maps of collected data sets of PH1.

of raw odometry and laser data. Since all environments are dynamic, we have different environment settings for each experiment. We examine two data sets with the most changes between the environments and use them to test localization and mapping.

We apply a standard SLAM approach [38] to generate the initial map m_0 based on the first data set, and the ground truth (i.e. $x_{1:t}$ and m_t) based on the second. It has to be noted that this approach does not usually suit dynamic environments [108] well. However, during *one* recording of a data set, only few dynamic changes occur. Therefore, the SLAM approach is a reasonable source for ground truth. The state transition model for each map is based on the structure of the environments. All three scenarios can be seen in Figure 3.3, 3.4 and 3.5. We omit the odometry data for clarity; a noteworthy fact is that it is generally very unreliable.



FIGURE 3.4: Ground truth maps of collected data sets of PH2.

To learn the parameter set Θ of the extended sensor model we use four separately recorded data sets. As we know the state transition model st , the initial map m_0 and the ground truth m_t and $x_{1:t}$, we can generate the training set $D = \{d_i \mid d_i = (c_{t-1}^i, c_t^i, st^i, z_t^i), \forall i \in [1, S]\}$ from each data set, S being the size of the data set. The goal is to compute Θ , so that the probability $P(D|\Theta)$ is maximized:

$$\arg \max_{\Theta} P(D|\Theta).$$

This problem can be solved by applying a maximum likelihood estimation (MLE) algorithm, given that there is enough data available. Despite that our data sets consist of millions of data points, we observe that some combinations of measurements, cell states and state transition dynamics rarely occur. Therefore, we assume a prior probability $P(\Theta)$ and utilize a posteriori probability (MAP) estimation [17]



FIGURE 3.5: Ground truth maps of collected data sets of LC.

TABLE 3.1: Learned parameters of the extended sensor model, denoted in %.

st	c_{t-1}	c_t	$P(hit c_{t-1}, c_t, st)$	$P(hit_o c_{t-1}, c_t, st)$	$P(miss c_{t-1}, c_t, st)$	$P(miss_o c_{t-1}, c_t, st)$
1	<i>occ</i>	<i>occ</i>	86.365	1.289	7.126	5.220
		<i>free</i>	7.955	8.721	71.494	11.830
	<i>free</i>	<i>occ</i>	14.223	10.720	72.709	2.349
		<i>free</i>	0.223	1.114	95.130	3.533
0	<i>occ</i>	<i>occ</i>	69.103	2.084	24.707	4.106
		<i>free</i>	34.575	0.550	7.251	57.626
	<i>free</i>	<i>occ</i>	7.190	43.211	48.979	0.619
		<i>free</i>	15.208	14.684	57.694	12.413

to compute Θ :

$$\begin{aligned} & \arg \max_{\Theta} \frac{P(D|\Theta)P(\Theta)}{P(D)} \\ &= \arg \max_{\Theta} P(D|\Theta)P(\Theta). \end{aligned}$$

With the assumption that $P(\Theta)$ is a Dirichlet Distribution, the optimal estimation can be obtained from the following calculation

$$\theta_{i,j} = \frac{m_{i,j} + \alpha_{i,j}}{\sum_i (m_{i,j} + \alpha_{i,j})},$$

in which $m_{i,j}$ is the number of samples with $\hat{z}_t = i$ and $(c_{t-1}, c_t, st) = j$, $\sum_i m_{i,j}$ is the number of samples with $(c_{t-1}, c_t, st) = j$ and $\alpha_{i,j}$ as the hyper-parameters which can be regarded as imaginary counts from the prior experience. The resulting estimates are denoted in Table 3.1.

In the following section, we report localization and mapping results, based on 10 runs for each scenario. Global scan matching [76] is used to provide the initial position of the robot.

3.5.1 Localization results

To verify the quality of localization, we consider maximal translation error and maximal angle error. We consider a run as a success, if the translation error is kept below 1.0 meter and the angle error below 25 degree. Otherwise we consider it as a failure. Firstly, we compare our approach (*DL*) with two static localization methods, namely, *AMCL* and *AMCL_{DF}*, that do not update the map at run time. *AMCL* is a standard localization method in the ROS package [84] which implements the popular Monte Carlo localization method described by Fox et al. [107]. *AMCL_{DF}* is the extension of *AMCL* with a distance filter [30]. All

methods use the same set of parameters for the motion model and the standard sensor model. For a fair comparison we increase the number of particles for *AMCL* and *AMCL_{DF}*, so that computational time required by every method remains at the same level at each iteration step (cf. Table 3.2). The threshold to allow map updates in our approach is set to 60% for all experiments.

Method	#Particles	Distance Filter	Scan Matching	Map Update
<i>AMCL</i>	1000	no	no	no
<i>AMCL_{DF}</i>	1000	yes	no	no
<i>DL_{simple}</i>	1000	no	no	yes
<i>DL</i>	500	yes	yes	yes

TABLE 3.2: Description of static and dynamic methods

All results are shown in table 3.3. As expected, localization with *DL* succeeds in all scenarios and all runs, whereas both static methods fail in most cases. Of course, *AMCL* has no way to incorporate dynamic influences. We can see that the distance filter helps, as *AMCL_{DF}* can deal with the dynamics in scenario PH1. However, as the scenarios become more complex, the success rate decreases. The reason is the well-known particle deprivation problem [114] which results in divergence of the particles in the second and third scenarios. To the contrary, our approach responds very quickly to changes and updates the map in time, so that the robot stays localized with the updated map.

We also evaluate an implementation of our approach which updates the map at run time, but has distance filter and scan matching disabled (*DL_{simple}*). We double the number of particles to achieve a fair comparison regarding computational time (cf. Table 3.2). It needs to be noted that without post-processing of scan matching, the map will be updated at every iteration step. This can lead to a divergence of the map. To overcome this problem, each particle should update and maintain its own map [108]. However, this will increase the load and usage of CPU and memory. As we can see from the results in Table 3.3, this is not necessary if we activate distance filter and scan matching.

3.5.2 Mapping results

To evaluate the mapping results we saved the updated map m_t at the end of each run. Accurate mapping should bring about accurate trajectories, especially

Scenarios	Method	Error (m)		Error (degree)		Success
		Mean	Var	Mean	Var	
PH1	<i>AMCL</i>	1.791	3.868	10.212	21.245	0/10
	<i>AMCL_{DF}</i>	0.138	0.051	0.638	0.523	10/10
	<i>DL_{simple}</i>	0.415	0.756	1.184	1.570	7/10
	<i>DL</i>	0.126	0.059	0.687	0.629	10/10
PH2	<i>AMCL</i>	0.750	0.957	6.337	9.286	1/10
	<i>AMCL_{DF}</i>	0.691	0.857	5.018	6.288	1/10
	<i>DL_{simple}</i>	0.160	0.080	1.381	1.307	10/10
	<i>DL</i>	0.165	0.054	0.615	0.528	10/10
LC	<i>AMCL</i>	10.107	23.054	22.442	48.222	0/10
	<i>AMCL_{DF}</i>	6.891	18.160	10.966	29.145	0/10
	<i>DL_{simple}</i>	3.144	4.282	3.356	7.112	0/10
	<i>DL</i>	0.110	0.096	0.602	0.883	10/10

TABLE 3.3: Comparison results of static and dynamic methods

in highly dynamic parts of the environment. Figures 3.6, 3.7 and 3.8 show the resulting maps with corresponding trajectories over all 10 runs for all approaches. As can be seen, the updated maps and trajectories have both converged to the ground truth in our approach, whereas other methods diverge in most runs (recall that *AMCL* does not update the map). The results of scenario PH2 also show that the resulting maps and trajectories of *DL* are more consistent and robust than the results of *DL_{simple}* despite that it also succeeds. This is also the case for the other scenarios. Altogether, these results suggest that our approach is very robust and consistent for different dynamic environments. Note that if we would use separated map updates for each particle, as described by Tipaldi et al. [108], we could expect similarly robust and consistent results for *DL_{simple}*. However, as this requires more resources, it may be more suitable to apply *DL*.

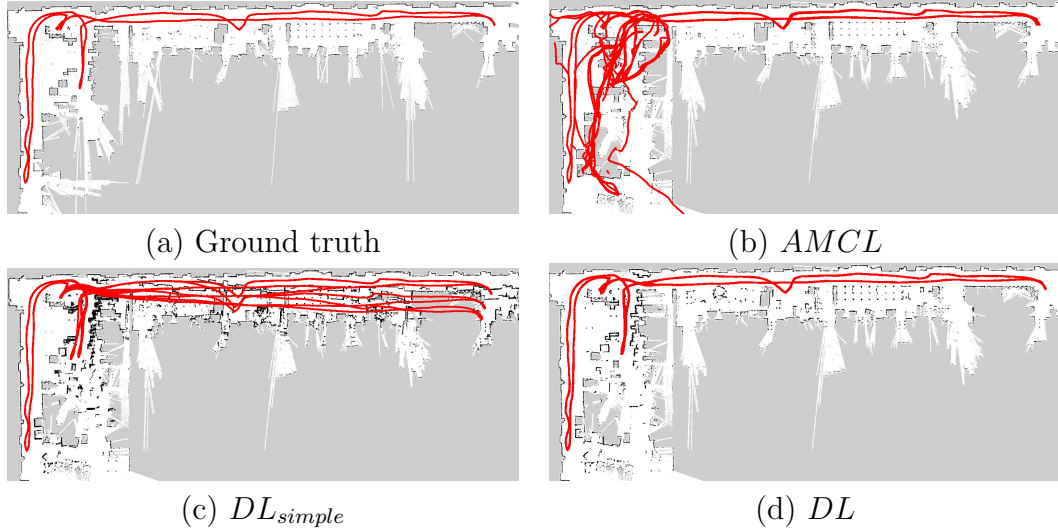


FIGURE 3.6: Mapping and localization results for PH1

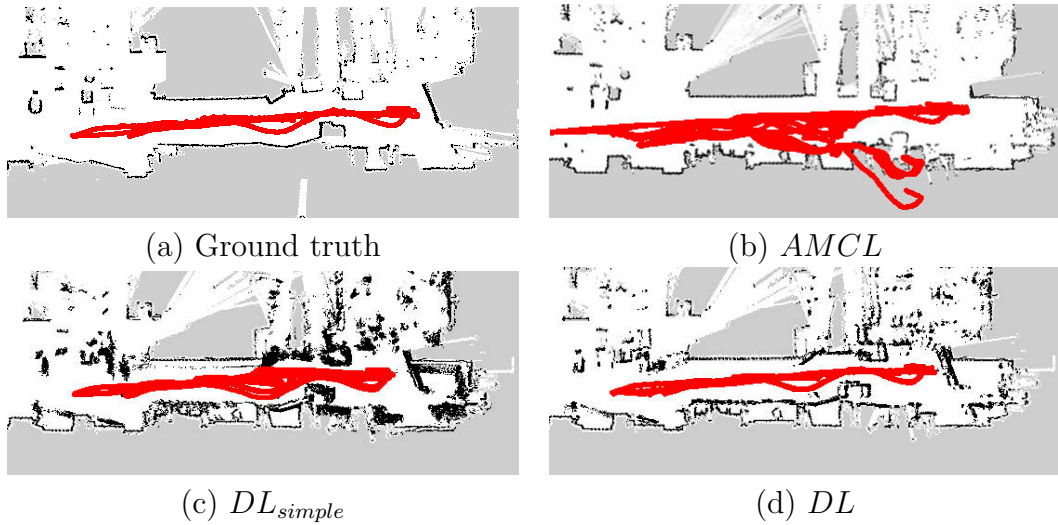


FIGURE 3.7: Mapping and localization results for PH2

3.6 Conclusion

In this chapter we present a dynamic localization approach which incorporates several previously introduced concepts. Similar to SLAM, we separate the problem and treat localization and mapping individually. By incorporating recent advancements on scan matching, we can give precise localization updates without consuming too many computational resources. Furthermore, we use a distance filter to incorporate dynamics of the environment directly into the sensor model, which leads to quicker convergence of the map update.

The empirical evaluation demonstrates the advantages of our approach. We learn the parameters of the sensor model and achieve a robust localization in several

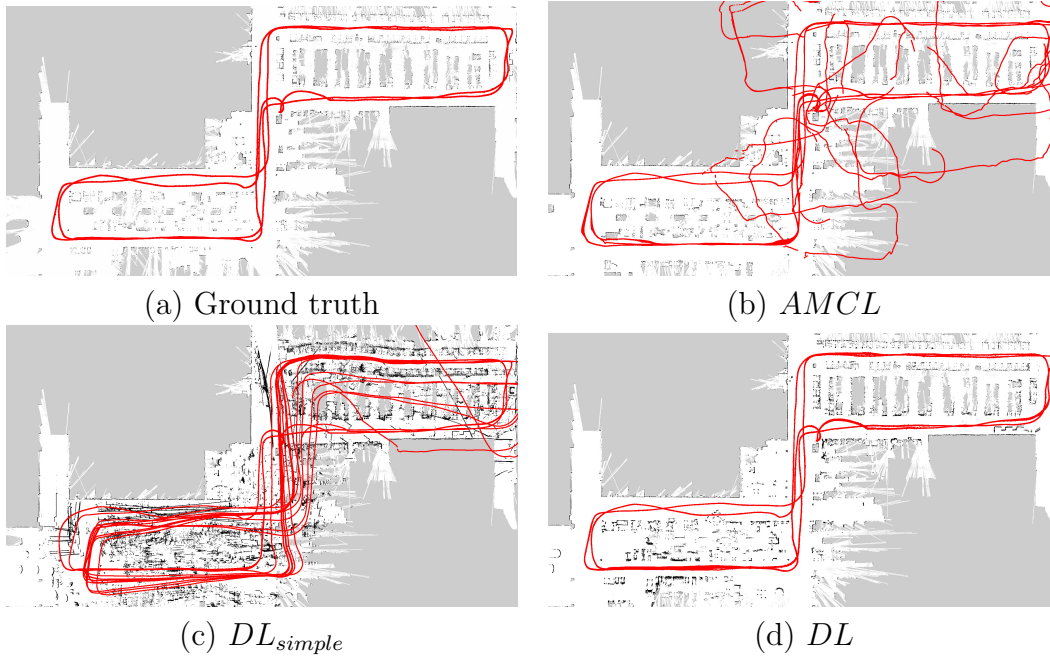


FIGURE 3.8: Mapping and localization results for LC

different real-world environments, all including dynamic influences. At the same time, the mapping stays consistent with the ground truth.

An important parameter for our approach is the threshold for map updates, which we determine empirically. Setting that threshold too high prevents an accurate representation of the real map and influences localization, while setting it too low can result in distorted maps, due to localization inaccuracies. It seems reasonable to aim for a threshold that brings about a good average matching score. Or we could merge the approach from Tipaldi et al. with ours. When the matching score drops below a certain threshold, we could associate each particle to its own map, until the matching score ascends again. Such threshold could be learned from data with the matching score as an evaluation function. Furthermore, all of our scenarios consist of environments with a clearly defined static structure, such as machinery and walls. While we exploit this information, there may be production halls which are completely dynamic, i.e., a very basic wall structure, no hallways or dynamic machinery equipment. The amount of static objects this approach requires remains to be an open question.

Another interesting topic is the dynamic localization in multi-robot scenarios. A basic consideration is to share the information of mapping results from multiple robots which can obviously improve the performance in the localization and mapping. A further consideration is to exchange robot trajectories and sensor data, so that a loop-closure could be identified as early as possible without requiring

robots to perform loops while executing their primary tasks. In such a case, a graph-based SLAM algorithm [67] in multi-robot scenarios [104] could be utilized to generate a global consistent map.

Chapter 4

Adaptive Road Map Optimization

4.1 Introduction

Besides the MRTA problem discussed in Chapter 2, another challenge in the multi-robot system is to efficiently coordinate robots in confined and cluttered environments.

For a single robot, the path planning or motion planning problem is a fundamental problem in the field of autonomous mobile robot navigation. The goal is to find out a plan in which a mobile robot is able to move from a start position to a goal position without collisions with any obstacles in its workspace defined in $\mathbb{R}^2$ or $\mathbb{R}^3$. The plan can be some kind of geometric paths or trajectories, or a sequence of motion commands. It might be very hard to find such a plan, if the rigid-body of robot is as complex as, for example, the “piano mover’s problem” [92].

In the field of robotics the obstacles in the workspace can be modeled either as combination of primitives (e.g. polygons, 3D-triangles) or as discrete representation of the workspace, for example, occupied cells. The rigid-body of a robot can be then mapped to a reference point (with the orientation) in the workspace, which is named as a configuration [66] in the workspace. The set of all configurations in the workspace is defined as configuration space $\mathcal{C}_{space}$. The set of all configurations that are collision-free with static obstacles is defined as free space $\mathcal{C}_{free}$. The path planning problem in the context of the definition of $\mathcal{C}_{space}$ is then defined as to find a path (or a trajectory) in $\mathcal{C}_{free}$ from a start configuration to a goal configuration.

The classical path planning approaches are road map based techniques, potential field based techniques and cell-decomposition based techniques [93]. The road

map based techniques [54] represent the connectivity of $\mathcal{C}_{free}$ as a network with line segments (e.g. Visibility diagram or Voronoi diagram). The path planning problem is then converted to the graph search problem. Many graph search algorithms can be applied to solve this problem. The widely used algorithms are perhaps Dijkstra's algorithm [20] and A* search algorithm [77], which is an extension of Dijkstra's algorithm by using heuristics to prove the time performance. Dijkstra's algorithm is complete and optimal. A* algorithm is complete too and by using admissible heuristics A* algorithm is also optimal.

The potential field based approaches represent the $\mathcal{C}_{free}$ as an artificial potential field [59] consisting of an attractive potential field generated by goal and a repulsive potential field generated by obstacles. An attractive force created by the attractive potential field pulls the robot towards the goal while a repulsive force created by the repulsive potential field push the robot away from the obstacles. A sum of two forces applied to a robot has the end effect that the robot moves toward the goal when it avoids collisions with obstacles. An advantage of potential field based approaches is that a path is not preplanned, but generated in real time. Therefore a robot can react very quickly to dynamic obstacles. But because of the local minimal problem, a robot can be trapped in a local optimum before it reaches the goal.

The cell-decomposition based approaches decompose the free space by cells (see one example, the occupancy grid map discussed in Chapter 3). The adjacent cells connect together and build a graph representing the free space $\mathcal{C}_{free}$. The path planning problem in context of the cell-decomposition is then a graph search problem. Because the graph can be more huge and complex than the road map, A* algorithms is the best choice for this problem.

From single robot to multi-robot system, the path planning problem is more challenging and difficult and in general is a Multi-Robot Motion Planning (MRMP) problem [78]. The MRMP problem can be described as follows: There are a team of mobile robots in which each robot has its own start and goal configuration. The problem is to find an individual plan for each robot in $\mathcal{C}_{free}$ while avoiding collisions with other robots.

The MRMP problem has been profoundly studied worldwide since 1980s and there are various approaches which normally can be divided into two categories: Multi-Robot Path Planning (MRPP) methods and Multi-Robot Motion Coordination (MRMC) methods. The MRPP methods are usually focused on explicit path planning, whereas in MRMC methods, a planning of a path for each robot is not always necessary, or only used to indicate the global direction of motion.

For instance, potential field based approaches, velocity obstacles based methods, and some swarm behaviors based approaches like flocking, schooling or formation-keeping. In this chapter we will focus on how to solve MRMP problems through explicit path planning methods, namely, MRPP methods. In next chapter, we will discuss another MRMC method inspired by swarm principles.

Most of MRPP methods can be roughly divided into two major categories: centralized and decoupled approach. In the centralized approach, a team of the robots is considered as a single composite system, and the classic path planning algorithm is applied in joint configuration space [5]. Centralized approaches are complete and optimal, but since the dimension of the joint configuration space grows linearly ($\mathcal{PSPACE}$ [43]) and thus the search space grows exponentially with the increasing number of robots, centralized approaches are not scalable and impractical for large problems.

In contrast to centralized approaches, decoupled approaches try to solve the problem independent [62]. Decoupled approaches are efficient and scalable, but they lost the completeness. This means that the solution is sup-optimal, or they cannot find a solution, even though there exists one.

Decoupled approaches can be typically divided into two broad categories: prioritized path planning [27] and path coordination [50]. In the prioritized path planning, robots are at first sorted by priority and then the path collisions are solved by searching a plan in a configuration time space for each robot in descending order in which lower priority robots treat the higher priority robots as dynamic obstacles. Furthermore, according to the timing of solving collisions, such approaches can be denoted as local repair approaches and cooperative approaches [95]. In the first case, all robots plan its individual path ignoring all other robots, then resolve emerged path collisions, if necessary, during runtime. In the other case, robots search paths with consideration of paths of all others at the beginning. Regardless of when collisions are solved, all of them are very sensitive to the priority scheme. In general, the local repair algorithms are more efficient than cooperative algorithms, since in cooperative algorithms, the complete search of all robots in the three dimensional configuration time space is more computationally costly. So the local repair algorithm is more suitable for real-time use. But the quality of the solution of local repair algorithm is lower, compared with cooperation algorithms.

In path coordination methods [50], first an individual path for each robot is generated without considering the paths of other robots. Next, a velocity planning method is used to tackle path collisions during the path execution by changing

robot's velocities (move slow, stop or move back). Such approaches are more or less suitable for the virtual agents in virtual environments. Because the exact control over robot's velocities is not easy in the real environment, the velocity planning method does not contribute very much on the real robot system. It's worth noting that if the robot can completely deviate from the planned path, the velocity planning method is actually a Multi-Robot Motion Coordination method. The velocity obstacles method [29, 113] is a good example.

As is mentioned, road map based approaches are popular for the single robot motion planning problem in static environments. However, for multi-robot applications and dynamic domains, road map based approaches are more challenging due to permanent changes in the environment.

In this chapter, we present a road map based approach called Adaptive Road Map Optimization (ARMO) to deal with such challenges. Firstly, an optimal road map is generated with considering environment constraints and the current demand for transportation tasks in the pre-processing phase. Next, each robot extracts its individual path from the road map, and an efficient decoupled MRPP approach will be used to coordinate robots during runtime. The road map will be recomputed after the changes of environment are detected. It is capable of adapting the road map in real time whenever the environment changes.

The next sections of this chapter is organized as follows: Section 4.2 describes some related works according to the road map based approaches. Section 4.3 illustrates our approach in detail. Section 4.4 shows the experiments and results of our approach. Section 4.5 concludes this chapter and indicates the future work.

4.2 Related Work

Lydia E. Kavraki et al. present an efficient path planning method based on so-called Probabilistic Road Maps (PRMs) for single robot in static environments [55]. This approach separates the planning in two phases: Learning phase and Query phase. In the learning phase, a graph called probabilistic road map by connecting the random sample of free space is constructed and stored. In the query phase, local connections from the start and goal configuration to the graph are built and a path will be found through searching over the graph using A* algorithm. This method can be applied to any type of robots with various degrees of freedom.

Kallman et al. present a method for single robot motion planning, employing the dynamic road map in changing environments [49]. The dynamic road map is

coupled with a grid-based cell decomposition of the workspace, and a road map of the free space is generated with the method of PRMs. For each cell in grid, a list containing all nodes and edges of the road map affected by this cell is pre-computed and stored. During runtime, occupations and liberation of cells are updated by tracking dynamic obstacles, and the affected nodes and edges of road map are considered as not safe and thus not allowed to be traversed during the graph search procedure. It is demonstrated that this method is faster and more capable of planning difficult motions in the 3D dynamic environment.

Silver et al. [95] introduce a decoupled Windowed Hierarchical Cooperative A* algorithm (WHCA) for multi-robot path planning problems that limit the search depth in configuration time space. This approach is a variant of the prioritized path planning method between the local repair algorithms and cooperative algorithms. Each robot plans only a partial path in the configuration time space with a fixed depth specified by the window, after which, it plans a path ignoring all paths of higher priority robots. Then, each robot begins to follow the partial path planned in configuration time space and when it is half-way on its partial path, the window is shifted forwards and a new search in configuration time space is started. The experiment with up to 100 robots suggests that the WHCA is more efficient than conventional prioritized path planning with the similar quality of paths (path length). However, their method is incomplete and there is no guarantee of the quality of the computed solutions.

Wang et al. [118] introduce a tractable algorithm for multi-agent path planning on grid map. At the beginning, individual paths are computed for each agent without considering paths of other robots. At runtime, no further path re-planning is needed and path collision is avoided by putting a blank travel idea, with the inspiration from sliding tile puzzles. Intuitively, an agent can move from a current location to the next location on its path, only if the next location is empty (so-called a blank). But if the next goal location is occupied by a lower priority agent, this algorithm tries to bring a blank to the next location which is similar to the blank that travels in a sliding tile puzzle. The researchers prove that a blank travel is always possible if the problem is a so-called *SLIDEABLE* problem. Thus this algorithm provides completeness guarantees on the problems that belong to *slideable* problem. But this algorithm is incomplete in the general case and there is also no guarantee of the quality of the solutions.

Luna et al. [68] propose a coupled fast algorithm called *Push and Swap* which can provide completeness guarantees for a general class of problems without any assumptions about the graph's topology. This algorithm employs two operations: a

“push” operation that allows robots to move towards their goals and a “swap” operation that allows two robots to swap positions without altering the configuration of all other agents. Compared to the conventional coupled A* approaches, this algorithm focuses its search on a necessary subset of operations, thus it is more efficient. Compared to decoupled algorithms, this algorithm provides completeness guarantee for a general class of problems that there are at most $n-2$ agents in a graph with n vertexes. Although the solution quality is noticeably better compared to WHCA, there are still no solution quality guarantees.

Bennewitz et al. [6] present an approach to optimize the priorities for prioritized path planning and test it in the real robot system. Since the quality of the solutions based on the prioritized path planning methods is very sensitive to the priority scheme, they perform a randomized and hill-climbing search in order to optimize the priority scheme and to minimize the overall path lengths. The search can be stopped at any time and return to its currently best priority scheme. They show experimentally that the number of failure of the path planning is reduced significantly and the overall path length also decreases remarkably with the use of their optimization of priority scheme. But there is neither completeness nor quality guarantees on the solution.

Velagapudi et al. introduce a distributed version of prioritized path planning for large teams of hundreds of robots [116]. They divide path planning into various iterations. In the first iteration, every robot plans a path simultaneously and independently and at the end of this iteration it transmits its path to lower priority robots. In subsequent iterations, every robot must re-plan its path, if the paths of higher priority robots from previous iteration have been received. At the end of each iteration, if a new path of a robot is produced by the re-planning, the robot sends this new path to lower priority robots. The whole process stops, if no path of higher priority robots from previous iteration is received. They show that this distributed version has similar quality to that of a centralized prioritized planner.

Berg et al. present a method based on the road map for multi-robot motion planning in dynamic environments [112]. The road map of free space is generated without considering dynamic obstacles in advance. Then, they use a two-level approach to find a trajectory in the road map. On the local level, trajectories on single edges of the road map are found by the depth-first search in a state-time space. On the global level, a near-time-optimal global trajectory is found by coordinating local trajectories. In contrast to our method which learns changes of the environment on-line, their approach discriminates between static and dynamic objects in advance, e.g., walls and robots, which might fail, when portions of the map have to be considered as dynamic as well.

Sud et al. develop an approach for path planning of multiple virtual agents in complex dynamic scenes [102]. They utilize first- and second-order Voronoi diagrams as a basis for computing individual agent paths. Then, they compute the local dynamics for each agent to follow the individual path based on a generalized potential field method. Their method is computationally efficient, but they do not focus on optimizing the global efficiency of the multi agent team.

4.3 Adaptive Road-Map Generation and Optimization

In our setting, there are transportation tasks D expected to be delivered among a set of fixed stations S by a team of autonomous robots $R = R_1, R_2, \dots, R_n$. Transportation tasks will arrive over time, so D is not static, but a function of time denoted by $D(t)$. For each transportation task $d^{kl} \in D(t)$ a robot is assigned to deliver a crate from a source station $k \in S$ to a destination station $l \in S$. We assume that the dynamic multi-robot task allocation problem has already been solved by adopting the DHHT method presented in Chapter 2, and hence we pay our attention to the MRMP as follows: Each robot has a start configuration $s_i \in C_{free}$ and a goal configuration $g_i \in C_{free}$, where C_{free} is the set of all configurations which are collision free given the footprint of the robot. Our MRPP problem is to compute for each robot $R_i \in R$ a trajectory $\pi_i : [0, T_i] \rightarrow C_{free}$ so that $\pi(0) = s_i$ and $\pi_i(T_i) = g_i$ which is free of collisions with the trajectory π_j of any other robot $j \neq i$. It should be noted that T_i denotes the individual path length of the robot R_i , and in our case the environment is dynamic, so C_{free} is a function of time denoted by $C_{free}(t)$.

According to the complexity of such MRPP problems, decoupled approaches are normally preferred for the large problem. In decoupled approaches, road map based approaches are more suitable than grid map based approaches for large problems, because the searching for solutions to road map is more efficient than that to grid map. Furthermore, the complexity of multi-robot collision solving would be much easier if robots are only restricted to the road map. For the same reason, the multi-robot navigation on the road map is more efficient in common case, and it is just similar to the highway for the high-speed traffic.

But as is mentioned in the introduction, a main drawback of road map based approaches is that it can not deal well with dynamic changes in the environment. Berg et al. [112] and Sud et al. [102] have already made some improvements to

avoid such drawback, but they are either not sufficiently flexible or not efficient enough. Therefore, we want to extend the conventional road map based approach to an efficient multi-robot navigation method which can adapt successfully to permanent changes in the environment.

For this purpose, we at first use the so-called *Dynamic Occupancy Grid Module* [71] to detect the changes of the environment and to integrate the changes to the grid map. Based on the up-to-date updated grid map, an optimal road map is automatically generated with considering environment constraints and the current demand of the tasks. In order to generate such optimal road map, all possible road segments connected all stations must at first be extracted from the Voronoi Graph [15] based on the up-to-date grid map. Afterwards, an optimal configuration of these segments is determined by a Linear Programming (LP) planner with respect to minimal travel costs and maximal compactness of the network. Finally, the road map is published to robots and each robot extracts its individual path from the road map. The path execution of robots is then guided with a decoupled navigation method.

Figure 4.1 depicts the overall system architecture and modules of the considered components. The localization module reports inconsistencies between sensor observations and the current grid map to the Dynamic Occupancy Grid Module. The updated grid map is published to the localization module of each robot, and to the Adaptive Road Map Planner as well. The Adaptive Road Map Planner computes an optimal road map configuration and publishes it to the local planner of each robot. The local planner extracts its own local path based on the road map and the plan is executed by the navigation module.

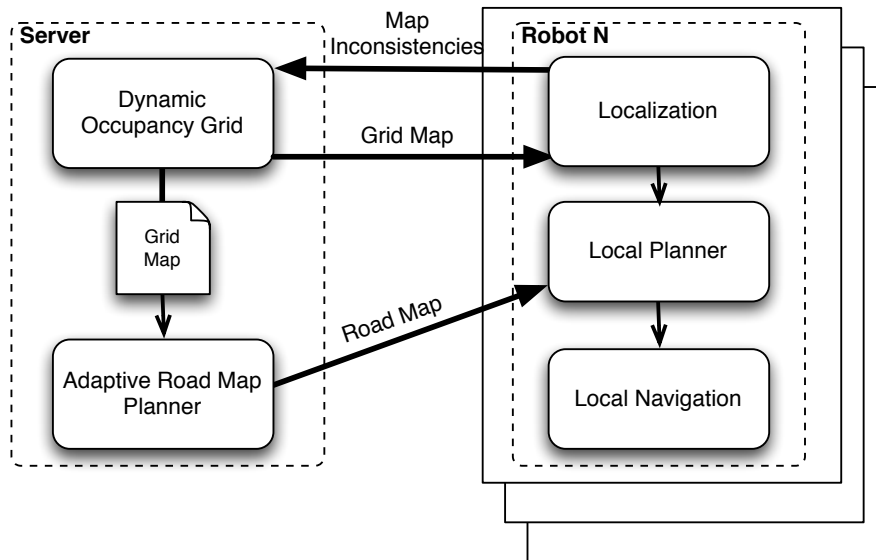


FIGURE 4.1: System Overview

It is noteworthy that the overview does not contain the mechanism for the multi-robot task allocation (MRTA), i.e., it does not assign robots to transportation tasks.

4.3.1 Detection and Integration of Dynamic Changes

In our approach, the environment is represented by an occupancy grid map [26, 73] which has been generated by an appropriate method for simultaneous localization and mapping (SLAM) in advance [37]. This grid map is used for both the localization and the generation of the road map. In order to adapt to the changes of the environment, this grid map should be updated when inconsistencies between laser observations and the grid map have been detected. In this chapter, we use the *Dynamic Occupancy Grid Module* [71] to update the occupancy grid map, and hence the standard occupancy grid map in this case is extended to the dynamic occupancy grid map.

In more concrete terms, a conventional occupancy grid map is combined with a Hidden Markov Model (HMM) that represents the belief in the occupancy and occupancy change probabilities of each cell in the grid. Standard occupancy grid maps are a special case of this representation in which the probabilities of change are zero. To measure the amount of changes in the environment, we count the cells that change their occupancy state between time t and time $t+k$ and divide it by the number of cells updated in this time interval. Whenever the percentage of changed cells is larger than a given threshold (e.g. 30%), a new road map is computed. In this case, the Voronoi Graph and connected network which connects all stations must be recomputed (see chapter 4.3.2). Figure 4.2 depicts the re-arrangement of the road map after local changes of the environment have been detected.

4.3.2 Computation of the Connectivity Network

In order to obtain an optimal road map, we first need to compute the Voronoi graph [15] based on the dynamic occupancy grid map. After that, we generate a connectivity network $\mathcal{C} = (V, E)$ based on the Voronoi graph, where the nodes $v \in V$ represent either station locations $loc(s)$ or crossings and edges $e \in E$ connect all stations and crossings on the map. The computation of $\mathcal{C}$ is carried out with the following three steps:

1. For each tuple $(i, j) \in \mathcal{S} \wedge i \neq j$, a set of alternative paths A_{ij} which connects station i and j on the Voronoi graph is determined.

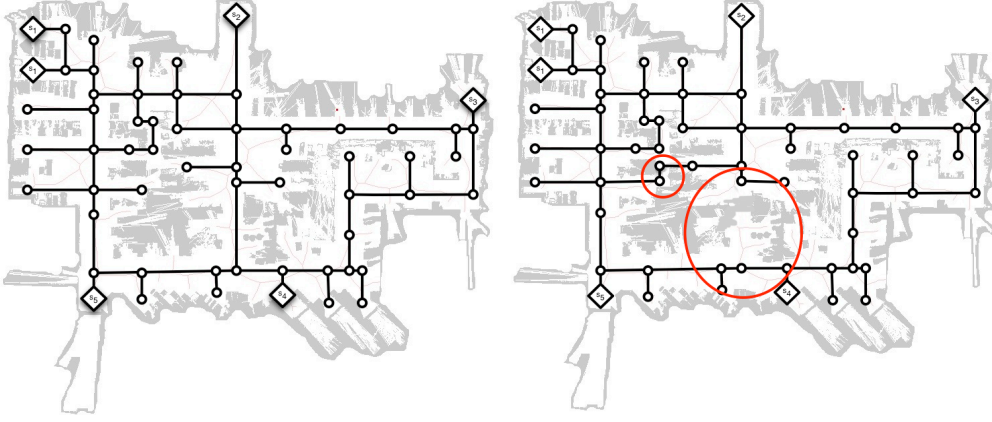


FIGURE 4.2: In industrial environments the map can locally change due to replaced objects, such as pallets, as well as gathering humans. Adaptive road map optimization facilitates the simultaneous navigation planning of large robot teams when taking these changes into account.

2. According to the method described in [16], we replace each A_{ij} by orthogonal straight lines (either horizontal or vertical) under the constraint that they have to be within a minimum safety distance to obstacles including the maximal extent of robots from their rotational center.
3. Add all straight lines to E . Merge parallel lines when the distance between them exceeds the double size of the robots.

Meanwhile besides station locations $loc(s)$, a new node is created for each crossing of edges and added to V . Finally, for each edge $e_{ij} \in E$, the maximal number of possible lanes w_{ij} and the time needed to cover this segment c_{ij} will be computed according to the distance to the nearest obstacle and length of the edge respectively.

4.3.3 Definition of the LP Problem

Based on the connectivity network $\mathcal{C}$, we want to find a road map configuration which minimizes the transportation cost with consideration of environment constraints and the current demands of stations, which is very similar to the *minimum cost flow problem* [2]. In the standard *minimum cost flow problem*, a certain amount of flow should be sent through a directed graph from source nodes to destination nodes. The flow running through an edge will cause a certain cost and the maximal amount of flow can not exceed the capacity of this edge. The goal is to find the way to send the flow through the graph so that the total cost is minimized. The *minimum cost flow problem* is a kind of optimization problems which can be

solved by linear programming. Therefore, we define our transportation problem with the help of the linear programming which is similar to the *minimum cost flow problem*.

More concretely, each time there exists a set of simultaneous *transportation tasks* $D(t)$, where each task $d^{kl} \in \mathcal{D}(t)$ is to deliver a package from a source station $k \in \mathcal{S}$ to a destination station $l \in \mathcal{S}$. Let $\mathcal{C} = (V, E)$ be a connectivity network which is computed as described in Section 4.3.2, where V is the set of nodes and E the set of possible edges. Besides a cost c_{ij} and the maximal number of lanes w_{ij} associated with an edge, we define the capacity u_{ij} of an edge as the maximal number of robots that can travel on this connection via a single lane at the same time without resulting in any congestion.

With the difference with the standard *minimum cost flow problem*, the edge is defined in both directions in our case, and hence the number of lanes in both directions between two nodes i and j is indicated by the decision variables y_{ij} and y_{ji} , respectively. For example, $y_{ij} = 1, y_{ji} = 2$ denotes a single lane connection from node i to node j and a double lane from node j to node i . In this case, the maximal number of lanes w_{ij} restricts all possible assignments to y_{ij} and y_{ji} , which means that $y_{ij} + y_{ji} \leq w_{ij}$. For example, if $w_{ij} = 3$, some of the possible assignments are $(0, 0)$, $(0, 1)$, $(1, 0)$, $(2, 1)$, $(1, 2)$, $(2, 2)$, etc. Note that the maximal number of lanes only depends on the space available in the real world, not on the directions of the edge, and thus $w_{ij} = w_{ji}$.

The decision variables x_{ij}^{kl} define the flow running through an edge ij due to the delivery from station k to station l . The total flow x_{ij} has to be greater than or equal to zero and below the maximal capacity $u_{ij}y_{ij}$, where u_{ij} is the capacity of a single lane and y_{ij} the number of activated lanes. In addition, we define $b^{kl} = b(d^{kl})$ as the required throughput rate, i.e., the amount of packages that have to be delivered per minute from a source node k to a sink node l . Finally, the problem can be formulated as follows:

$$\text{Minimize } \sum_{(i,j)} \sum_k \sum_l c_{ij} x_{ij}^{kl} + \sum_{(i,j)} u_{ij} y_{ij} \quad (4.1)$$

subject to:

$$\sum_{j:(j,i)} x_{ij}^{kl} - \sum_{j:(i,j)} x_{ji}^{kl} = \begin{cases} -b^{kl}(i) & (i = k) \quad \forall i, k, l \\ b^{kl}(i) & (i = l) \quad \forall i, k, l \\ 0 & \text{otherwise.} \end{cases} \quad (4.2)$$

$$y_{ij} + y_{ji} \leq w_{ij} \quad \forall (i, j), \quad (4.3)$$

$$x_{ij}^{kl} \geq 0 \quad \forall (i, j), k, l, \quad (4.4)$$

$$\sum_k \sum_l x_{ij}^{kl} \leq u_{ij} y_{ji} \quad \forall (i, j), k, l \quad (4.5)$$

$$\sum_{j:(i,j)} x_{ji}^{kl} \leq C_{max} \quad (i \neq k \wedge i \neq l) \quad \forall i, k, l \quad (4.6)$$

Eq. (4.1) minimizes over the total travel costs and the physical space occupied by the road network. Eq. (4.2) enforces the flow conservation in the network, i.e., the summed flow from all incoming edges $j : (j, i)$ and all outgoing edges $j : (i, j)$ has to be equal $-b^{kl}$ if i is a sink, b^{kl} if i is a source, and zero otherwise. Eq. (4.3), Eq. (4.4), and Eq. (4.5) are constraining the maximal number of lanes, minimal and maximal flow, respectively. Finally, Eq. (4.6) ensures that the total flow through crossing nodes does not exceed the maximal crossing capacity C_{max} which depends on the spacial size of crossings, i.e., how many robots can be located there at the same time. Note that transportation tasks for which the node operates as source or sink have no influence on the capacity.

The formulation above can efficiently be solved by linear programming solvers (e.g. CPLEX), when we define the decision variables x_{ij}, y_{ij} by continuous values and rounding up the y_{ij} from which the road map can directly be constructed then. Furthermore, we yield for each transportation task d^{kl} a subset of edges from the road map that has positive flow assignments $x_{ij}^{kl} > 0$. These quantities are directly utilized by the local planner (see Section 4.3.4) to extract individual robot paths.

4.3.4 Local Planner

Given an optimal road map and a transportation task d^{kl} , each robot extracts its individual shortest path on the road map and follows this path while coordinating locally at crossings with other robots.

First of all, the local planner builds local connections from start and goal configuration to the road map and searches for the shortest path over the road map by using a A*-like algorithm. To expand a node i in the road map during the search, the local planner performs random sampling over all outgoing edges weighted according to their normalized flow values x_{ij}^{kl} . If there exists only one edge with $x_{ij}^{kl} > 0$ for node i , the edge is expanded directly.

After that, a local navigation module is applied to coordinate robots on the road map. At a lane, a robot, like a convoy, just follows the previous robot. At a crossing, a distributed prioritized path planning method [116] is applied to avoid collisions with other robots. Instead of a static priority used in [116], the priority in our approach is dynamic.

Every time when a robot arrives at a crossing, it begins to count the “process time” every 100 ms (started with 0) and check if the overall capacity of the outgoing edge is already exceeded or not. There are two possibilities when the overall capacity of the outgoing edge is already exceeded. One is that there is already a higher priority robot at the crossing and it wants to move to the same outgoing edge. Another is that the number of robots on the outgoing edge has already reached the maximal capacity of this edge. Either possibility being the case, the robot must wait at the crossing and continuously count its “process time”. As soon as there are no such robots at the crossing, a distributed prioritized path planning is started for a group of robots, which includes a new arrival and the robots which are already at the crossing. The priority is determined by the “process time”: the longer the “process time”, the higher the priority. In case of the robot has the same “process time” with others, the priority is determined by the IDs of robot, which are pre-defined and unique in our approach. It is noteworthy that it is possible that there are more than one new arrival joints in the path planning procedure, when several new arrivals are entering the crossing at the same time. The robots that are already in the crossing will not stop counting its “process timer” until they leave the crossing, since a path planning procedure can be triggered at any time by some new arrivals.

Because the prioritized path planning is not complete, it is possible that some robots can not find a plan. The robots which have no valid plan must wait for a few seconds and then trigger a new round of path planning again. That will not lead to a traffic congestion in the crossing when there is a failure of path finding of some robots. Because in the first round of the path planning, at least one robot (with highest priority) can find its plan to pass through the crossing due to the property of the road map optimization. After that, if a robot already has a valid plan in the previous round of the planning procedure, it can not be blocked by

new arrivals with lower priority in the next round of the path planning procedure. Actually the waiting will only happen to the new arrival which has not yet entered the crossing. The robots that already have a valid plan will never be trapped during the passing of the crossing, and the maximal capacity of the outgoing edge has already been checked before the robot enters the crossing. Therefore, at any time at least one robot is able to pass through the crossing and reach its outgoing edge.

4.4 Experiments and Results

This approach is tested in several different environments simulated by a simulator Stage [115]. Figure 4.3 depicts these environments of the experiments presented in this section. The PLANT map has a size of $51m \times 56m$, the ASE map a size of $94m \times 82m$, and the KNO map a size of $88m \times 43m$. On each map we defined locations of loading stations: 8 on PLANT, 16 on ASE, and 8 on KNO.

In the section below, a comparison between our approach ARMO and a decoupled prioritized planning (PRIOR) approach from Berg and colleagues [111] will be presented with large robot teams (up to 100 robots). In prioritized path planning, trajectories of robots are planned iteratively after a pre-defined priority scheme. When the i 'th robot is planned for, the trajectories of the $i-1$ robots planned previously are considered as dynamic obstacles. Berg and his colleagues define the query distance as the distance for each robot to reach its goal configuration on the shortest path while ignoring the other robots. In order to minimize the maximum of arrival times, priorities are assigned according to this distance: the longer the query distance, the higher the priority assigned to a robot. This prioritized path planner is complete under the assumption that start and goal configurations of each robot are so called garage configurations, i.e., configurations that are not part of C_{free} of any other robot. The method avoids efficiently the intractable computation of $n!$ possible priority schemes, however, it requires at least R sequential calls of the motion planner.

We utilized the Stage software library [115] to simulate a large robot team and the environment. One advantage of Stage is that it allows the building of simulation worlds directly from occupancy grid maps generated from real environments.

We generated 100 transportation tasks for each map which are handled by 20, 50, and 100 robots during different runs. Table 4.1 provides the results of comparing ARMO with PRIOR on different maps with different numbers of robots. We

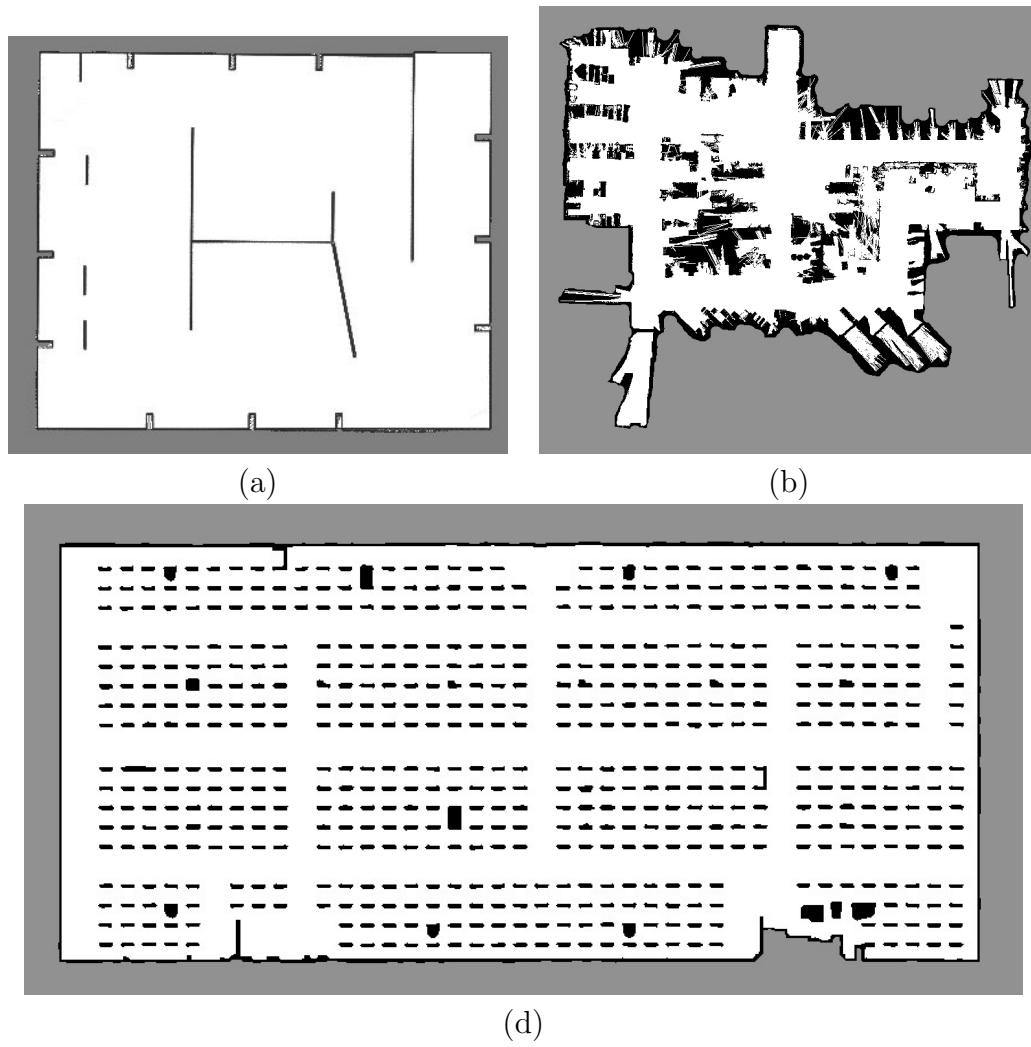


FIGURE 4.3: Grid maps utilized for experiments: (a) the PLANT map generated from a simulated environment, (b) the ASE map generated in a real logistic center, (c) the KNO map generated in a large distribution center.

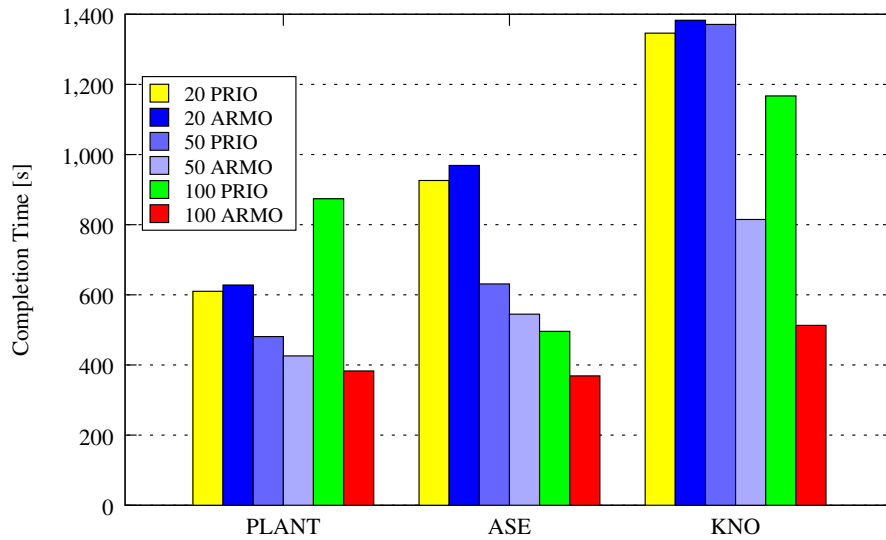


FIGURE 4.4: Comparing the CTime of prioritized planning (PRIOR) with adaptive road map optimization (ARMO).

measure the number of conflicts (C) of the optimal path in C_{free} with trajectories of the other robots. In the case of ARMO, these are the situations in which a robot has to wait for other robots before entering a segment, and in the case of PRIOR, these are the situations in which robots has to plan around a conflicting path of a higher prioritized robot. Furthermore, we measure the average velocity of all robots (avg. v) and the total time needed by all robots to complete all tasks (CTime). As can be seen from Table I and Figure 4.4, while leading to slightly longer completion times for small robot teams, ARMO reduces significantly this time when the team size increases. This is also reflected by the number of conflicts and the average velocities of the robots. The prioritized path planning minimizes the final completion time following a heuristically determined order, whereas the LP-based planning in ARMO minimizes the global flow of robots, leading to a more efficient distribution of the robots over time.

Map	#Robots	Method	#C	avg.v(m/s)	CTime (s)
ASE	20	ARMO	1432	0.47	969
		PRIOR	2142	0.43	926
	50	ARMO	5040	0.37	545
		PRIOR	10625	0.28	631
	100	ARMO	8307	0.3	369
		PRIOR	16983	0.2	496
PLANT	20	ARMO	1471	0.42	628
		PRIOR	1346	0.38	610
	50	ARMO	5563	0.34	426
		PRIOR	11601	0.25	481
	100	ARMO	15145	0.22	383
		PRIOR	107700	0.21	874
KNO	20	ARMO	506	0.38	1383
		PRIOR	5638	0.35	1346
	50	ARMO	2951	0.31	815
		PRIOR	70799	0.16	1371
	100	ARMO	7729	0.29	513
		PRIOR	102836	0.11	1167

TABLE 4.1: Comparing prioritized planning (PRIOR) with adaptive road map optimization (ARMO)

The computation times of both methods are measured in seconds on an Intel DualCore running at 2.13 GHz. We measure PRIOR with 50 robots and got an average computation time of 0.030 ± 0.03 on PLANT, 0.05 ± 0.04 on ASE, and 0.1 ± 0.17 on KNO, and with 100 robots 0.10 ± 0.08 on PLANT, 0.13 ± 0.08 on ASE and 1.1 ± 0.7 on KNO. ARMO requires the road map computation $t1 = 0.9 + t2 = 0.6$ on PLANT, $t1 = 0.82 + t2 = 0.84$ on ASE, and $t1 = 1.2 + t2 = 10.3$ on KNO,

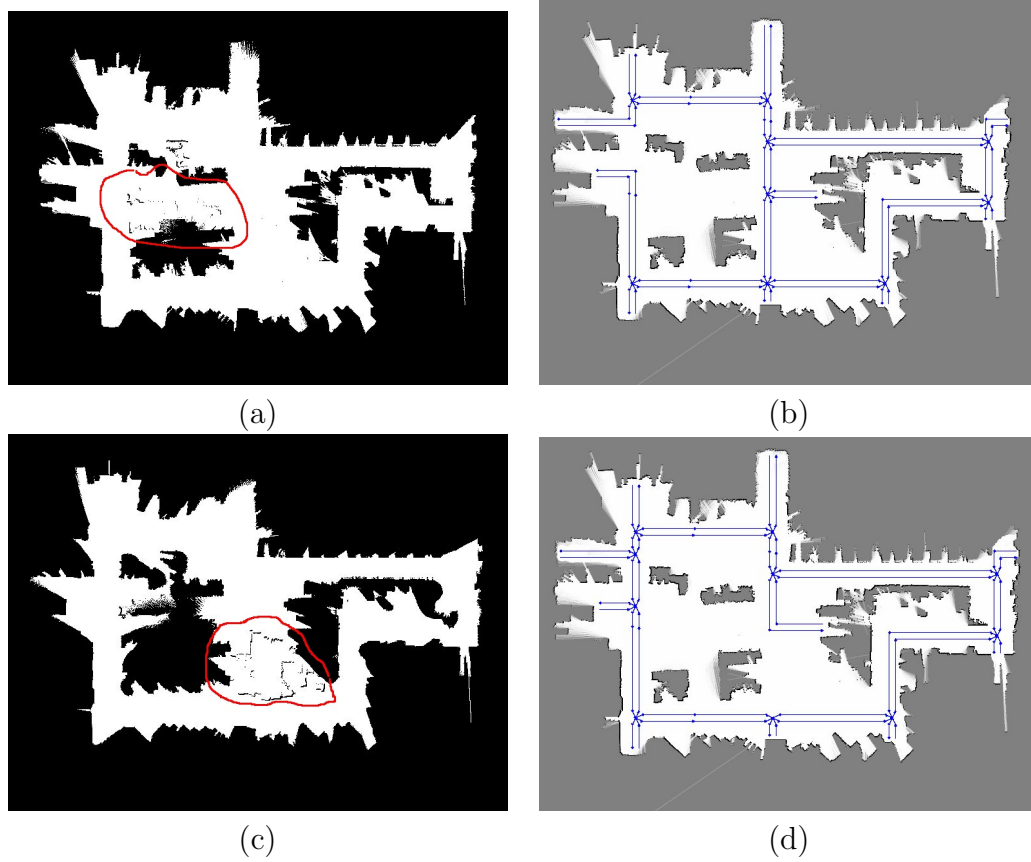


FIGURE 4.5: Adjustment of the road map according to dynamic changes in the map (a,c) source of disturbance and (b,d) resulting modifications reflected on the road map.

where t_1 is the time for extracting the connectivity network, and t_2 the time for solving the LP problem. Within each planning cycle, ARMO needs on average only 0.002 on PLANT, 0.004 on ASE and 0.01 on KNO for any number of robots. In summary, the number of robots has nearly no effect on the computation time needed by ARMO, however, we detect a significant growth of the time needed by prioritized path planning. On the contrary, ARMO requires much more time to generate the road map when the environment is very large and complex, such as the KNO map, which however needs only to be performed at low frequency, i.e., when the environment is significantly changed (see Section 4.3.1).

4.4.1 Adaption in Dynamic Environments

We also evaluate ARMO with respect to dynamic changes of the grid map. For this purpose we modify the ASE map step wise by adding successively obstacles updated in the map by the dynamic occupancy grid approach. Figure 4.5 depicts two snapshots taken at successive points in time. As can be seen, the road map adjusts to the changes at the cost of higher completion times. For 100 robots

the completion time increases from 378s (no modifications) to 410s (first modification) and 420s (second modification). We perform several more experiments for evaluating the adaptivity of our approach. Also after changing the distribution of transportation tasks among the stations, the road map is dynamically adjusted by removing or adding links among the stations. Note that in this case only the LP solver is restarted without re-computing the connectivity network.

4.4.2 Real-World Experiments and Results

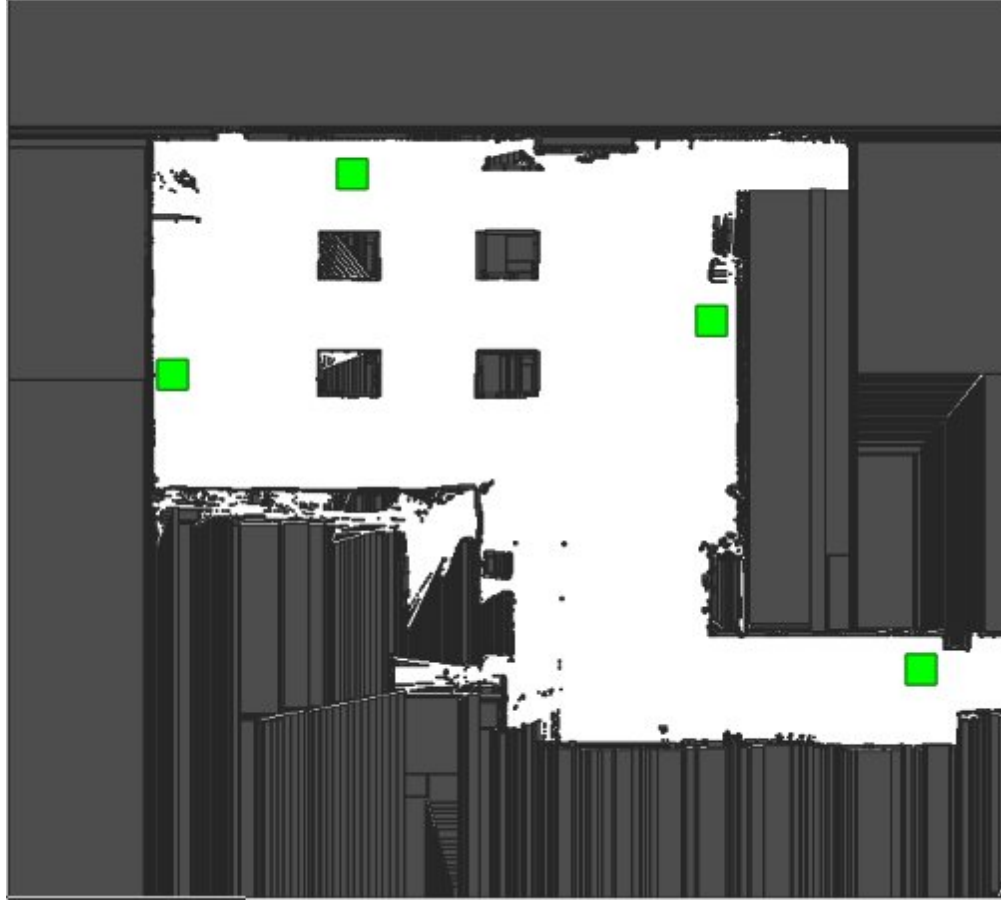
We also conducted real-world experiments with four KARIS robots (see Chapter 6) for the comparison between pairs of a ARMO or PRIOR method combined with a task allocation method, DHHT or ContractNet. To put it in a simple way, we compare four pairs of methods, each of which consists of one MRMP method from PRIOR or ARMO and one MRTA method from DHHT or ContractNet.

The number of arrived packages at loading station is modeled by a Poisson distribution with parameter λ . Each station is characterized by the average number of packages which arrive within one time unit. Without loss of generality, at the end of each experiment in the same map with same number of robots, the number, destination and sequence of the generated transportation tasks are totally identical and thus the completion time between different pair of methods is comparable. To fit the different numbers of robots, the λ varies from 2 to 4.5 packages/min. Like assembly lines in the real world, arrived packages are kept in a virtual queue attached to each station. Robots load the first arrived box from the queue and deliver it to its destination.

Similar to experiments in the simulation, we measure the completion time, average velocity of all robots and the number of conflicts to reflect the efficiency of the different pair of methods. Furthermore, we also measure the average efficiency, fairness of all stations and the worst-case time-delay of package deliveries to illustrate the difference between different task assignment methods.

Due to the limited capacity of batteries we set the maximum execution time up to 45 minutes for each experiment. During all experiments the robots are running more than 6 hours autonomously in total without causing collisions or dead-locks. The environment of the experiments is the robot experiment area in the University of Freiburg and the Figure 4.6 (a) depicts the map of the testing environment with four stations marked in green. Four KARIS finish transportation tasks for stations where station 1 is placed comparatively far from the others and in no task is it set as the destination. Thus, this station is disadvantaged because of its location and

the setting of transportation tasks. Figures 4.6 (c) and (d) show the robots and environment during the experiments in the experimental area.



(a)



(c)



(d)

FIGURE 4.6: Real world experiments in the robot experimental area of University of Freiburg. (a) Snapshot pictures captured from Stage. (b) and (c) show four robots and the environment during the experiments.

Table 4.2 shows the results of this experiments. As can be seen, with the same MRTA method, the ARMO method is always more efficient than the PRIOR method, e.g., the number of delivery of ARMO+DHHT is 111, and the number of delivery of PRIOR+DHHT is only 79. But due to the low density of robots in the environment, the superiority of DHHT compared to ContractNet cannot be

Real-World experiments					
M	#C	v(m/s)	Eff:	#Deliv.	WCTD(s)
ARMO+DHHT	294	0.48	0.41	111	280
ARMO+ContractNet	385	0.39	0.49	128	2667
PRIOR+DHHT	929	0.44	0.30	79	640
PRIOR+ContractNet	410	0.39	0.48	122	2577

TABLE 4.2: Results of Real-World experiments of map Building.78

clearly reflected by the results. In the case of ContractNet the robots just move among three stations which are located close to each other. However, in the case of DHHT the robots must serve station 1 continuously and thus waste much time on the road to station 1. That is the reason why the average efficiency of stations with DHHT is not as high as that with ContractNet. Nevertheless, by comparing the average velocity, the fairness and the worst-case of deliveries, we can still find some competitive advantages of DHHT over ContractNet.

As can be seen in Figure 4.7 which shows the standard deviation of fairness, DHHT always attains an obviously better degree of fairness. Figure 4.8 depicts efficiency of stations over time. The efficiency of stations with DHHT method (Figure 4.8(b)) converges to a fair distribution, while the difference of efficiency of ContractNet method (Figure 4.8(a)) is huge: the efficiency of station 3 is always 1.0, whereas station 1 is always 0.

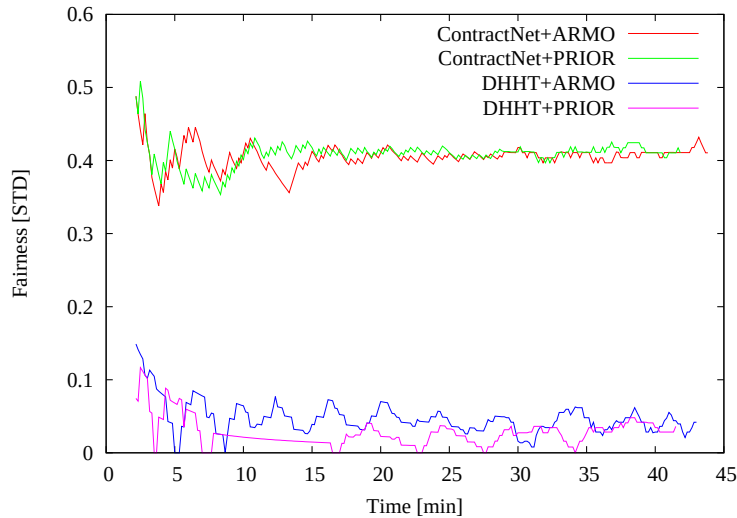
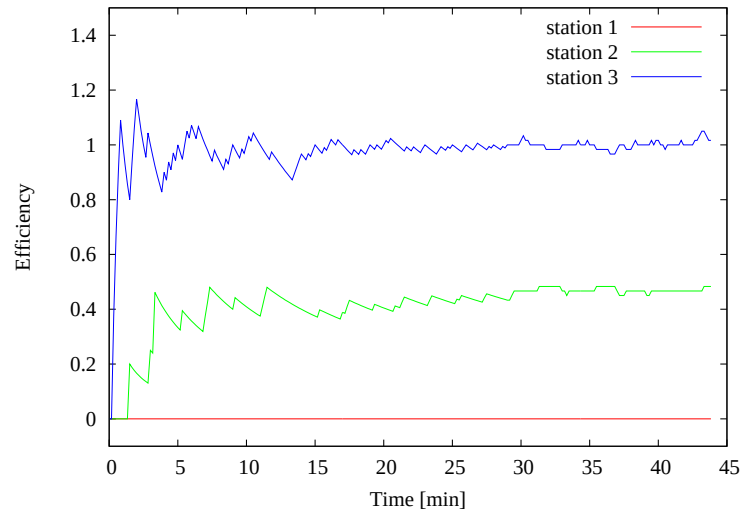
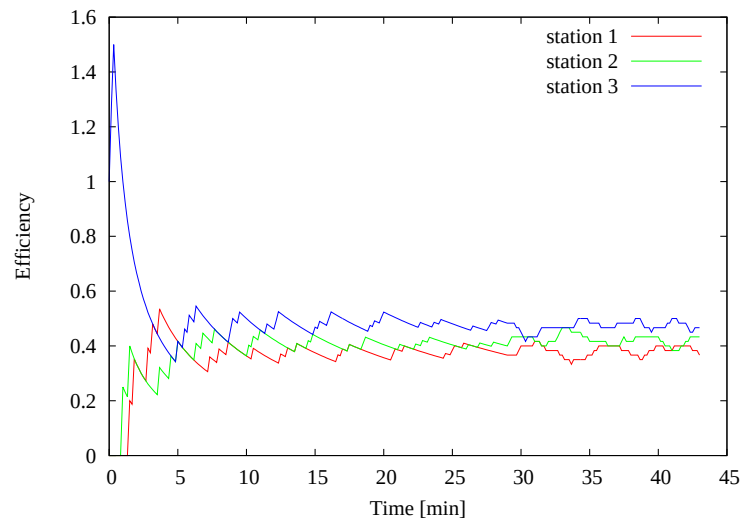


FIGURE 4.7: Standard deviation of fairness over time. Each color denotes a pair of methods.



(a) ContractNet+ARMO



(b) DHHT+ARMO

FIGURE 4.8: Efficiency of 3 stations over time. Each color denotes a single station. Station 4 has no outgoing tasks.

4.5 Conclusion

In this chapter we propose a road map based approach called Adaptive Road Map Optimization (ARMO) to deal with multi-robot path planning (MRPP) problems. Our approach generates an optimal road map with taking into account environment constraints and the current demands of stations in a pre-processing phase. Then, the robots extract individual paths from the road map by a local path planner and cooperatively navigate themselves by a local navigation module. The optimal road map will be recomputed only when changes of environments have been detected or the demands of stations have been changed. Furthermore, the computation time of generated an optimal road map depends mainly on the complexity of the environment rather than on the number of robots. Therefore, our approach is able to be used for the large robot team (up to 100 robots) in dynamically changing environments.

We conduct several experiments in simulation and on the real robot system to test our approach under different conditions. The experimental results have shown that the ARMO approach leads to more efficient multi-robot navigation than conventional prioritized based decoupled path planning methods.

Chapter 5

Swarm-Intelligence: Behavior-based Collision Avoidance

5.1 Introduction

For virtual agents, many Multi-Robot Path Planning (MRPP) methods mentioned in Chapter 4 can be used to solve the Multi-Robot Navigation Problem. But due to the imprecise control and many uncertainties in real environments, accurate path tracking is not easy to realize. Thus, MRPP methods do not suit real robot systems quite well. By contrast, in Multi-Robot Motion Coordination (MRMC) methods, a global path is not necessary, or is just used to indicate the global direction of motion. The collision will be avoided locally by applying immediate reaction to changes in surrounding areas. In the context of MRMC, swarm intelligence principles based approaches are an important technique to deal with the local collision avoidance. Therefore, in this chapter we concentrate on a collision avoidance method inspired by swarm intelligence principles.

Swarm Intelligent [9, 45] refers to the design and development of algorithms or distributed problem-solving systems inspired by the collective and emerging behavior of biological systems, such as social insects (e.g. ants, bees, etc.) or animals (e.g. birds and fish). The collective and emerging behavior of such biological systems is triggered automatically by relatively simple rules followed by every individual without any central control system. Local communications between members can be achieved in a direct way or an indirect way. Although each such insect or animal is not considered as intelligent, the whole societies of them can show some

self-organization behavior which can be regarded as some kind of collective intelligence. For example, a group of birds forming a special pattern when they are in flight is called the flocking behavior; a group of fish swimming in the same direction in a coordinated manner is called the schooling behavior; a group of goats instinctively gathering together as a group and moving from place to place to flee a predator is called the herd behavior; a group of bees or ants scattering in different directions simultaneously to find the best food source is called the foraging behavior.

Through biological research on such collective behaviors, we can understand the underlying mechanisms of such biological systems and develop similar algorithms or artificial systems. For instance, the classical Particle Swarm Optimization (PSO) algorithm [56] inspired by the flocking and schooling behavior is a population based optimization method for optimization of continuous nonlinear functions. The Ant Colony Optimization (ACO) algorithm [21] inspired by the ant foraging behavior is a probabilistic method used to solve path planning problems. The Artificial Bee Colony (ABC) algorithm [51] inspired by the foraging behavior of bees is a population based heuristic algorithm for solving constrained optimization problems. The Bees algorithm [80] inspired by the foraging behavior of bees is a population based search algorithm for solving continuous optimization problems. There are many other algorithms and almost all of them share information among members via direct or indirect communications. Furthermore, the multiple members can be parallelized easily, so that these algorithms are usually more suitable for the large, discrete, continuous or constraint optimization problems, like traveling salesman problems, large scheduling problems, large network routing problems, clustering problems and image processing problems.

The applications of swarm intelligence principles to robots is called swarm robotics [93, Chapter 40.4][90]. Swarm robotics focus on how to develop artificial distributed robotic systems which attempt to achieve a desired group-level collective behavior without relying on any form of centralized control. In such systems, the individual robot normally has too limited ability to solve a complete problem on its own, but with a huge number of such robots they can solve the problem together or better. The collective behavior of the robots results from local interactions between the robots and between the robots and the environment in which they act. Local interactions between robots can rely on the direct communication or the indirect communication, for instance, via pheromone. The most important advantage of a swarm system is that the system is fault tolerant, flexible and scalable.

Firstly, because of the distributed architecture, swarm systems are able to deal well with the failure of one or more individuals. Some individual crashes will not indicate the failure of the whole system. With this property, swarm systems are highly suitable for applications with dangerous tasks which imply a high risk of losing some individuals. Secondly, swarm systems can also deal well with the changes of environments because of its self-organization property. The collective behaviors are generated from simple rules and local interactions instead of global information of the whole environments. Thus, swarm systems are perfectly suitable for applications in which sudden changes of environments are very difficult or even impossible to be predicted. Last, swarm systems are fully scalable because of its local sensing and communications property. Individuals interact only with local communications or even without direct communications. So, swarm systems are particularly suitable for very large problems and some particular tasks such as underwater or planetary exploration in which an infrastructure for communications does not exist in advance.

The design and development of a swarm system focus on how to define behaviors and rules at the level of individual robots so that the desired global collective behavior at the level of group can be achieved [10]. According to the different global collective behaviors, the swarm systems can be roughly divided into four categories: spatially-organizing behaviors, collective decision behaviors, navigation behaviors and other collective behaviors [1]. Spatially-organizing behaviors focus on how to organize and distribute robots and objects in space, such as aggregates, patterns, chains and structures of physically connected robots. Collective decision-making behaviors focus on how robots influence each other when they make decisions, such as consensus achievement and tasks allocation [60, 82]. Navigation behaviors focus on how to coordinate the movements of robots, such as collective exploration, coordinated motion and collective transport. Some other collective behaviors that can not fall into the three categories above build up the last category, other collective behaviors.

In the context of the Multi-Robot Motion Coordination (MRMC) problems, many navigation behaviors based approaches (see the section 5.2) have already been developed and used for real robot systems in spacious environments. But few of them perfectly fit applications in which robots must move in a structured, complex and constrained environment. For example, it is imaginable that there will be many chaos if two group of robots run reciprocally through a narrow corridor or a door. In this case, it should be better if some robots can just wait a moment for other's passing or choose an alternative path.

Based on these considerations, we propose a behavior-based multi-robot collision avoidance (BBMRCA) method which is essentially based on the swarm intelligence principles. In addition, we also combine a MPPP method with this method to make it more suitable for structured and crowded environments. In short, a path for each robot is computed without consideration of any other robot and all robots execute their paths simultaneously. If a path collision occurs during execution time, a situation specific behavior is automatically selected to avoid the collisions that the robot might temporarily deviate from the original and optimal path. Behaviors are selected according to specific traffic rules that directly emerge from the context. The path will be re-planned when the avoiding behavior fails or a deadlock is detected. Re-planning will also be triggered whenever the path is completely blocked by unknown obstacles, which can take place when, for example, the real environment deviates too much from the initial state.

The remainder of this chapter is organized as follows. In Section 5.2 some related works related to MRMC and swarm systems are described. In Section 5.3 the method for multi-robot collision avoidance is depicted in detail. In Section 5.4 results from experiments are presented and we finally conclude in the section 5.5.

5.2 Related Work

Sperati et al. [98] present an effective collective navigation in an unknown environment for a swarm of robots. The robots with limited sensory range move back and forth between two distant target areas and are located within a rectangular arena surrounded by walls. The navigation behavior at the level of the team is synthesised through a simple evolutionary algorithm used to optimize the parameters of a feed-forward neural network controller. They evaluate their approach in a simulator and show an interesting dynamic chain formation behavior. The robots self-organized a dynamical chain that merges and grows with two rows of robots which are moving in opposite directions. However, this approach is not tested on real robots or extends to a cluttered environment with obstacles.

Fiorini et al. [29] present a method for the motion planning of a robot in dynamic environments. They propose a definition of the velocity obstacle that represents the set of robot velocities that would result in a collision with given obstacles which move at each individual given velocities. The avoidance maneuvers are generated by selecting robot velocities with consideration of robot dynamics and actuator constraints outside such velocity obstacles. The trajectory from start to goal is computed by searching a tree of feasible avoidance maneuvers at discrete time

intervals. They demonstrate this method for disk robots and an automated vehicle successfully. But in the multi-robot scenario, when each robot does not take into account that other robots will also have the same strategy, the resulting motion may cause undesired side effects like oscillations. To prevent oscillations, Berg et al. [113] extend this method by taking into account that other moving objects react to the robot in the same manner. In their work, the avoidance maneuvers are generated by selecting a new velocity which is just the average of robot's current velocity and a velocity that lies outside the other robot's velocity obstacles. With this improvement, they demonstrate safe and oscillations-free navigation for more than 1000 agents in the virtual environments.

Ducatelle et al. [25] present a collective navigation behavior for swarm robots based on mobile ad-hoc network communications. The robots only have a limited communication range and want to move to one or more targets in a cluttered or uncluttered environment. The *navigation information* about a target spreads throughout the mobile ad-hoc network with the help of a so-called *navigation table*, in which the *navigation information* is stored and maintained and broad-casted by each robot. A robot that wants to move to this target makes use of the *navigation information* to move from robot locations towards the target. In this work the researchers study two different experimental setups: a single robot moves while the other robots act as beacons to guide it, and all the robots move between two targets. In the second experiment, the robots organize themselves into a robust a dynamical chain, similar to the work from Sperati et al. [98], but with completely different methods. They test their methods both in a simulator and on real robots and show that collective navigation behavior has a preference for short paths.

Turgut et al. [110] present a flocking behavior for swarm robots with the help of a so-called *virtual heading sensor* which allows each robot to sense the relative headings of other robots. With heading information and distance information (obtained by an infrared sensor) of other robots, the robots are able to obtain coherent coordinated motion and obstacle avoidance without being aware of a global goal direction. The authors evaluate the performance of flocking behavior by using different metrics and test it both on real robots and in the simulator. They show flocking behavior with a small group of real robots named "Kobot" in a closed constrains area. They also evaluate their method in a simulator with 1000 robots and show robustness against a big noise. This is the first true implementation of flocking behavior with real robots.

Ferrante et al. [28] also propose a flocking behavior for swarm robots with a novel method that does not need an explicit alignment rule. The robots use only attraction and repulsion rules that depend on relative distance and bear neighboring

robots to compute the forward and angular speed. The robots do not need to sense the heading directions of other robots, which is necessary in the work of Turgut [110]. Thus this method could be used on a very large swarm of robots which have only relatively simple hardware equipments. Furthermore, they show that their method is able to navigate robots in the absence of an informed robot that knows the global goal direction. Their method is evaluated by using different metrics and tested both in a simulator and on real robots. The real robot is called “foot-bot” and it is developed during the Swarmanoid project [23].

Kushleyev et al. [61] describe the architecture and algorithms to coordinate a team of quad-rotors in known three-dimensional environments. The micro quad-rotor has a size of 21cm diameter and a weight of 73 gram with on-board attitude estimation and it can operate autonomously with an external localization system. Due to the small size and weight, the micro quad-rotor has a high control agility and can fly in tight formation. They demonstrate with 20 quad-rotors how a team performs a tight formation flight and how four groups of four quad-rotors fly through a window.

Kernbach et al. [57] show a project SYMBRION (Symbiotic Evolutionary Robot Organisms) funded by European Commissions. The project focuses on “evolvability, dependability and artificial evolution for such robot organisms based on bio-inspired and computational paradigms ([57, p. 2])”. In this project, the swarm robots can dock with each other to share energy and computational resources. With the self-assembling property, the swarm robots can dynamically aggregate into a single symbiotic multi-robot organism and collectively interact with the world through a variety of sensors and actuators. They demonstrate that the swarm robots form a 3D structure (like a bridge) to collectively pass a barrier (see Figure 5.1). This project is an open-source and an open-hardware.

Rubenstein et al. [87] present a low-cost robot Kilobot (see Figure 5.2) to provide a testing platform for research on swarm algorithms. A Kilobot robot uses two sealed coin shaped vibration motors for locomotion, a infrared transmitter/receiver for communication and distance sensing, a three-color (RGB) LED to show internal state and a Lithium-Ion battery for 3-10 hours’ use. The robot has only the size of a coin and cost 14 dollars, which makes it possible to test collective algorithms up to hundreds of robots. Additionally, a large Kilobot team can be easily controlled by a single user, which is hardly possible for many other real robot teams. They demonstrate some popular swarm behaviors such as foraging, formation control, and synchronization with 25 Kilobots.

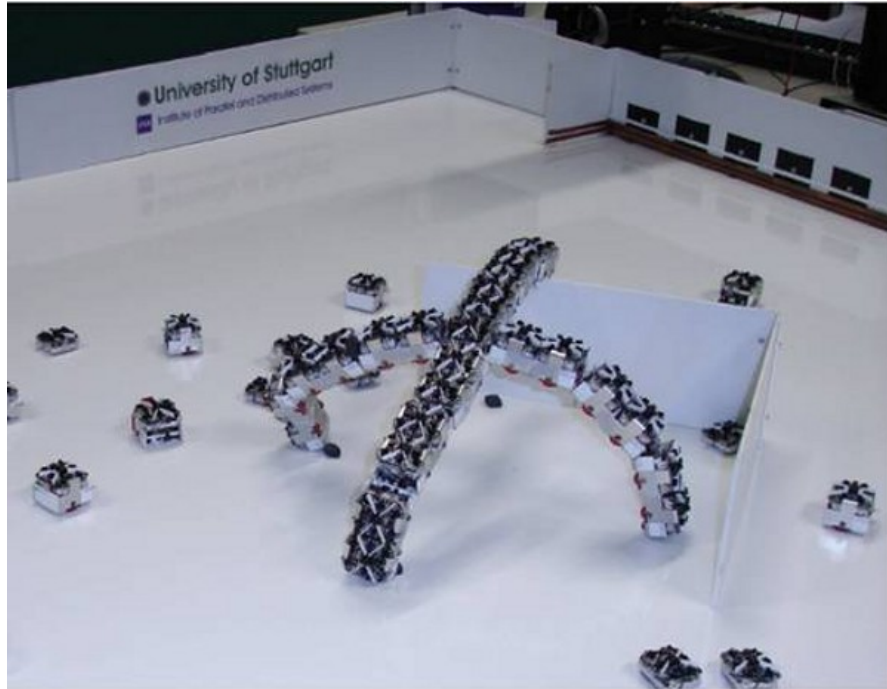


FIGURE 5.1: The swarm robots of project SYMBRION form a 3D structure to collectively pass a barrier (<http://zool33.uni-graz.at/artlife/SYMBRION>).

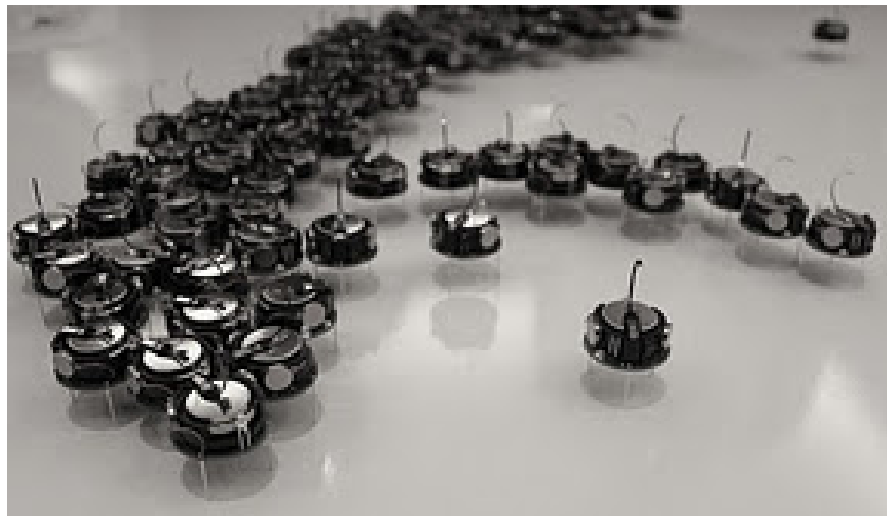


FIGURE 5.2: A Low Cost Scalable Robot System Kilobot (<http://www.eecs.harvard.edu/ssr/projects/progSA/kilobot.html>).

Dorigo et al. [22] show Swarmanoid project [23] in a movie which was the champion of the AAAI 2011 video competition. The Swarmanoid project, as the successor project to the Swarm-bots project [24], is a Future and Emerging Technologies (FET-OPEN) project funded by the European Commission. The Swarmanoid system consists of three different robot types (see Figures 5.3): “foot-bots are small autonomous robots specialized in moving on both even and uneven terrains, capable of self-assembling and transporting either objects or other robots; hand-bots

are autonomous robots capable of climbing some vertical surfaces and manipulating small objects; eye-bots are autonomous flying robots which can be attached to an indoor ceiling, capable of analyzing the environment from a privileged position to collectively gather information inaccessible to foot-bots and hand-bots ([23, p. 1]). They demonstrate successfully the integrated search and retrieval behavior in a real-world environment with a Swarmanoid robot team. The team could find a shelf and retrieve the book with no prior knowledge of where the book is.

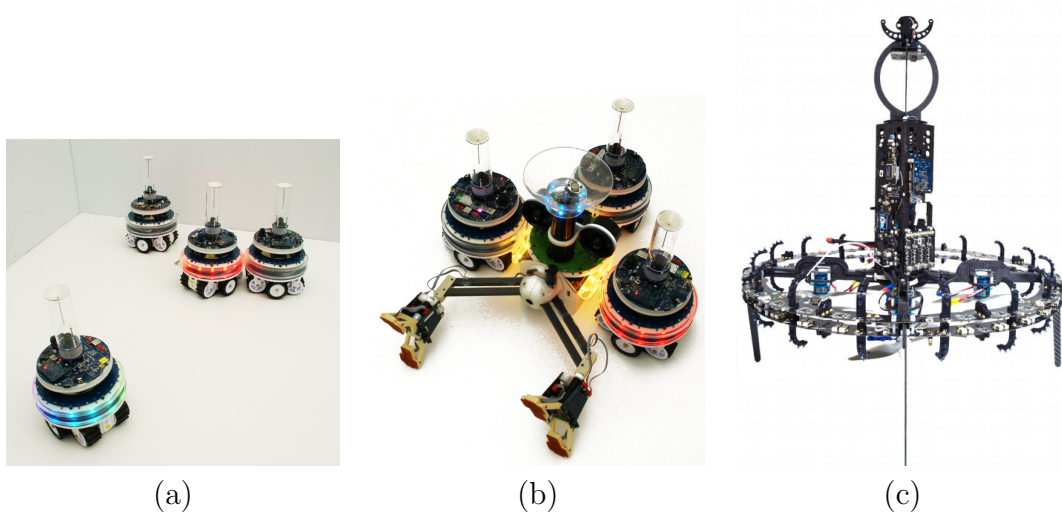


FIGURE 5.3: The Swarmanoid robot system consists of three different robot types: (a) foot-bots, (b) hand-bots and (c) eye-bots (http://www.swarmanoid.org/swarmanoid_hardware.php).

5.3 Behavior-based Collision Avoidance

In this section, we describe the behavior-based multi-robot collision avoidance (BBMRCA) method in detail. At first, the problem is formally defined in 5.3.1. Then, in 5.3.2 we describe how the behaviors and rules at the level of individuals are defined to achieve the global navigation behavior of the group. After that, in 5.3.3 we describe how a possible deadlock is detected and handled so that the completeness guarantee can be given for a subset of MRMP problems.

5.3.1 Problem Formulation

Similar to the last chapter, we consider the problem of coordinating the execution of transportation tasks by a team of autonomous robots between a set of fixed

stations $\mathcal{S}$. For each transportation task $d^{kl} \in \mathcal{D}(t)$ a robot has to be assigned to finalize the delivery by transporting the corresponding crate from station $k \in \mathcal{S}$ to station $l \in \mathcal{S}$. We assume that the task allocation problem has been solved (e.g. by DHHT as shown in Chapter 2), and hence restrict our attention to the problem of solving the MRMP problem as defined below. Let $\mathcal{R} = \{R_1, R_2, \dots, R_n\}$ be the set of n robots navigating simultaneously on a two-dimensional grid map. During planning, each robot has a start configuration $s_i \in \mathcal{C}_{free}$ and a goal configuration $g_i \in \mathcal{C}_{free}$, where $\mathcal{C}_{free}$ is the subset of configurations in which robots can take on without colliding with static obstacles. For each robot $R_i \in \mathcal{R}$ a path $\pi_i : [0, T_i] \rightarrow \mathcal{C}_{free}$ will be computed by executing the A* algorithm on a grid map with obstacles grown according to the outer radius of the robot footprint. It is noteworthy that T_i denotes the individual path length of robot R_i . If R_i does not have any potential collision with other robots while following path π_i , T_i is equal to the final path length. In case of a path collision between R_i and R_j , R_i or R_j has to take a detour or hold its position to avoid the collision. We denote by Π_{ij} the increased cost (in our implementation increased time) to avoid a collision between two robots R_i and R_j either due to the taken detour or induced wait steps. Then, T_i is increased according to either the additional path length or the wait steps. The problem is to develop a decoupled collision avoidance system to minimize the overall costs for avoiding collisions during the entire transportation process of all robots given by

$$\min \sum_{(i,j)} \Pi_{ij}.$$

5.3.2 Swarm Behaviors and Traffic rules

It is very hard to find a optimal solution of such problems. Hatzack and Nebel [40] shown in their work that scheduling the moves is $\mathcal{NP}$ -hard. Because of this, we do not attempt to solve this problem directly or exactly, but try to approximate an optimal solution by reducing the overall costs for collisions avoidance as much as possible. To achieve this goal, we invest in development of behaviors and rules at the level of individuals which can help robots avoid collisions within a certain little group efficiently. The basic idea is that all robots try to keep following the originally planned optimal paths as precisely as possible while avoiding collisions, so that each robot will keep the cost of detours as low as possible. Since we do not do any global optimization, the solution is not optimal. But we will show with the experiments that the quality of our method is much better than some conventional decoupled MRPP methods and similar to the ARMO method provided in Chapter 4.

As mentioned above, we are assuming that the task allocation problem has already been solved and each robot $R_i \in \mathcal{R}$ obtains a job assignment. For each robot a path will be computed with A* algorithm on the 2D occupancy grid map and the paths are executed simultaneously by all robots. During this process, robots are constantly exchanging information about their position, velocity, path and status whenever they are within the direct communication range. The robots are synchronized not with a global clock but with local messages and communications. This means that the robot does not broadcast the message but only sends it to its direct neighbors.

With the help of a so-called *FrontAreaCheck* function, each robot checks continuously whether other robots are situated within its way. The front area is defined by a rectangle as shown in Figure 5.4. The origin of this area is the current position of the robot. The orientation of the front area depends on the current position of the robot and the next local goal from the path to be reached, whereas the current orientation of the robot is ignored. In other words, if the robot rotates at its current location, the orientation of the front area will not change. The critical area is located in the front area and corresponds to the drive channel needed by the robot.

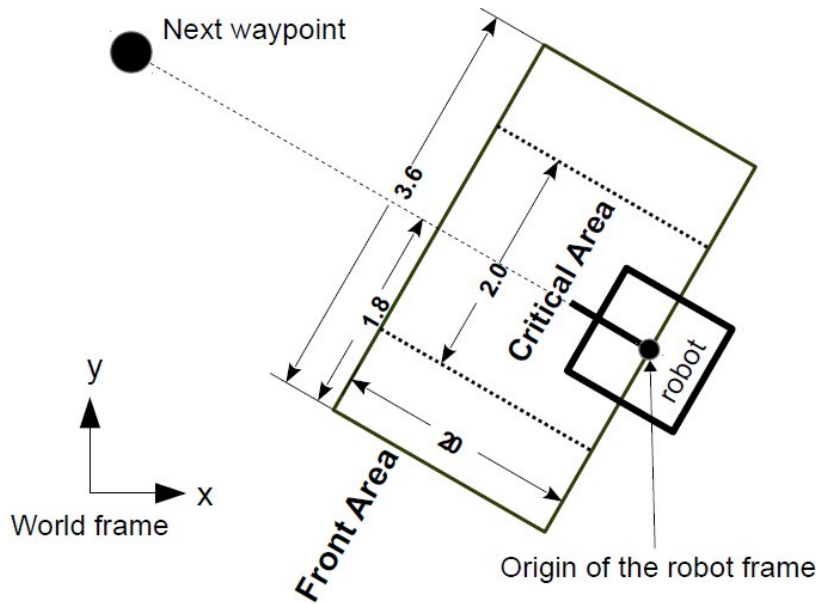


FIGURE 5.4: Definition of the front area and the critical area.

The closest robot to the origin in the front area will be selected by the front area check and the robot in the critical area will be preferably selected. It means that, if any robot is located in the critical area, the robots which are not in the critical area will be ignored.

5.3.2.1 Definition of behaviors

The behaviors designed to solve the multi-robot navigation problem are listed as follows:

1. *FollowWayPoint*: The robot follows the next local goal one by one in the path until the robot reaches the final goal station. After loading or unloading a crate at the station, the robot will plan a new path to the new goal and execute *FollowWayPoint* again.
2. *Avoid*: The robot drives round the other robot which is its partner, or an unknown obstacle which is considered as its partner. When the partner is not in the front Area any more, this behavior is then completed and the robot will execute *FollowWayPoint* again.
3. *Exchange*: For a head-on collision, two robots pass each other and they are partners of each other. When the partner is not in the front area any more, this behavior is completed and the *FollowWayPoint* behavior will be started again.
4. *GoThrough*: For a side collision, this robot is going through the intersection, the other has to wait until this robot passes. After passing through the intersection, the behavior is completed and the *FollowWayPoint* behavior will be executed again.
5. *Dock*: The robot reaches the docking region of a station and starts to dock with the station.
6. *WaitKeepDistance*: The robot waits for a partner and keeps a certain distance to the partner. When the partner leaves its front area, the behavior is completed and the *FollowWayPoint* behavior will be executed again.
7. *WaitForGoThrough*: When the partner is just carrying out the *GoThrough* behavior, this robot must wait and if necessary, make the intersection free for the partner to go through. If the partner stops his behavior or leaves its critical area, the robot will change to do *FollowWayPoint* again.
8. *WaitForDocking*: The robot must wait when another robot is just docking at the same station.

The *FollowWayPoint* behavior uses the data from the laser range finder (LRF) in order to avoid the collision with unexpected obstacles. The LRF data is also

used by the *Avoid* and *Exchange* behaviors to search the most suitable direction in which the robot should preferably move. In behavior *Exchange*, robots have to reach a consensus on a common avoiding side with the help of the LRF data and robot information. Note that in the *Avoid* behavior, the robot treats the other robot as its partner, but it is not necessary that the other robot also treats this robot as its partner. For example, a robot can use the *Avoid* behavior to circumnavigate a robot which is just waiting for a third one.

In this chapter we focus on intra-logistics problems and therefore two particular behaviors *Dock* and *WaitForDocking* are needed to handle the Loading and unloading process. When a robot is docking with a station and another robot arrives nearby, the latter should stop at a certain distance from the station and wait until the former leaves. When no robot is docking with the station, the robot should check whether other robots arrive and are ready to dock at the same time. Here we use a simple strategy, first come first served, and in case of the same arrival time, the robot with a higher priority will dock at first. The priorities are defined according to the robot IDs which are unique in our system. The end effect is that only one robot can dock with a station at a time.

The behaviors can be divided into two subclasses: one is so-called *Active* behavior and the other is so-called *Wait* behavior. As the name suggests, the subclass of *Active* behavior includes all behaviors (*FollowWayPoint*, *Avoid*, *Exchange*, *GoThrough* and *Dock*) in which the robot is in movement and the subclass of *Wait* behavior includes all behaviors (*WaitKeepDistance*, *WaitForGoThrough* and *WaitForDocking*) in which the robot stops. The execution of a particular *Active* behavior is controlled by a local reactive controller. In the reactive controller, the global path provides an attractive force that pulls the robots toward the local goal, and the obstacles detected by the LRF provide a repulsive force that pushes robots away from the obstacles. The obstacles might be other robots or some unknown and unexpected obstacles such as human workers or crates. The local controller can react very quickly with the dynamic changing in the environment, so it can provide a certain safety guarantee for human workers. But in the case of a traffic jam, for example, the road totally blocked with unexpected obstacles, the execution of a behavior may fail. If the execution fails, the reactive controller will report a failure and the robot will try to find an alternative path to the goal to avoid a deadlock. The deadlock detection and path re-planning will be discussed in detail in Section 5.3.3.

5.3.2.2 Definition of the Traffic rules

During the execution of behaviors, every robot checks continuously the front area for a possible new path collision which consists of rear-end collision, head-on collision and side collision. Once a new path collision is detected, the robot will select a new behavior according to the traffic rules. The traffic rules for each behavior are defined as follows.

1. FollowWayPoint: Algorithm 1.
2. Avoid: Algorithm 2.
3. Exchange: Algorithm 2.
4. GoThrough: Algorithm 3.
5. Dock: When a path collision happens, the robot will keep moving towards the goal instead of selecting a new behavior. In this case, the other robots will automatically make way for it.
6. WaitKeepDistance: Algorithm 4.
7. WaitForGoThrough: Algorithm 4.
8. WaitForDocking: In stead of changing the behavior, the robot will move back or move into a less crowded area when another robot is too close to it. However, the robot will stop WaitForDocking and execute FollowWayPoint again when it is too far away from the station because of the avoiding movement.

The function *SolveActiveAndWaitCollision* 5 is used to decide whether the robot should circumnavigate another robot or not. Simply put, in case other robots have the same goal station or are also within the critical area, the circumnavigation has to be delayed. Otherwise, if no other robot within the front area is doing an *Active* behavior, the circumnavigation can proceed. More detailed information about this algorithm is given in Algorithm 5.

The function *lowOrder* provides a dynamic priority scheme that determines the priority of robots according to the distance to the last goal station. The robot with a shorter distance to the last goal station has a higher priority. In case of the same distance, the priority is determined by the IDs. The idea within this function is that if a station is crowded with many robots, the robot closest to the station should first start to leave the crowded region. Additionally, with the

Algorithm 1: SolveFollowWayPointCollision

Data: the robot in the front area
--

Result: select a new behavior

begin

switch *behavior of the robot* do

case *FollowWayPoint*

if *rearEndCollision* **then**

WaitKeepDistance;

else if *headOnCollision* then

Exchange;

else if *sideCollision* then

Execute GoThrough or WaitForGoThrough depend on the situation;

case *Avoid* $\vee$ *Exchange* $\vee$ *GoThrough*

if I am the robot's partner then

Cooperate with the robot with a proper behavior;

else

```
| WaitKeepDistance;
```

case *Dock*

if *pathCollision* **then**

| Avoid;

else

```
| WaitKeepDistance;
```

$$\text{case } WaitKeepDistance \vee WaitForGoThrough \vee WaitForDocking$$

```
| SolveActiveAndWaitCollision(robot); // Alg.5
```

dynamic priority scheme, robots will have a chance to exchange the priority during runtime, which will reduce the possibility of deadlocks.

The function *noOtherActive* is used to check whether other robots in the front area are executing an *Active* behavior. The idea behind this function is to prevent additional possible collisions, when other robots in the same area are already engaged in avoiding collisions.

5.3.3 Deadlock and Re-plan

As is stated above, in case of a failure that executes the behaviors, the robot will plan a new path to the goal. For this purpose, a virtual wall will be built between the current position of the robot and its local goal in the path in the search space

Algorithm 4: SolveWaitKeepDistanceCollision**Data:** the robot in the front area**Result:** select a new behavior**begin** **switch** *behavior of the robot* **do** **case** *Avoid* $\vee$ *Exchange* $\vee$ *GoThrough* **if** *Avoid me* **then**

Cooperate with the robot with a proper active behavior;

case *Dock* **if** *pathCollision* **then**

Avoid;

case *WaitKeepDistance* $\vee$ *WaitForGoThrough* $\vee$ *WaitForDocking* **if** *Wait for me* **then** /* Deadlock */ **if** *lowOrder(robot)* **then**

Drive around the robot with a proper active behavior;

else **if** $\neg \text{sameGoalStation} \wedge \neg \text{inCriticalArea} \wedge \text{noOtherActive}(\text{robot})$ **then**

Drive around the robot with a proper active behavior;

Algorithm 5: SolveActiveAndWaitCollision**Data:** the robot in the front area**Result:** select a new behavior**begin** **switch** *behavior of the robot* **do** **case** *WaitKeepDistance* $\vee$ *WaitForGoThrough* **if** *I am not the partner of this robot* **then** **if** $\neg \text{sameGoalStation} \wedge \neg \text{pathCollision} \wedge \text{noOtherActive}(\text{robot})$ **then**

Avoid;

else

WaitKeepDistance;

else

Cooperate with the robot with a proper behavior;

case *WaitForDocking* **if** *sameGoalStation* **then**

WaitKeepDistance;

else

Avoid;

expanded by A^* . In this way, A^* will avoid using the same local goal in the original path and find an alternative path that does not go through the virtual wall. The failure of different behaviors is defined as follows:

1. *FollowWayPoint*: If the robot detects unexpected and unknown obstacles that blocks the entire front area, but no other robots are located in the front area.
2. *Avoid*: If the robot can not find any free space in the front area to execute this behavior and there is no other robot excepts its partner in the front area.
3. *Exchange*: If both the robot and its partner can not find free space to execute this behavior. The robot with lower ID will re-plan its path and the robot with higher ID will shift to *WaitKeepDistance* behavior.
4. *GoThrough*: The execution time exceeds 30.0 sec.

We assume, that the *Dock* behavior is always executable given the domain of intra-logistics and thus there is no definition about the failure of the *Dock* behavior.

With the alternative path re-planning, robots can prevent themselves to fall into the failure of the path execution. But it is still possible that robots run into deadlocks when they wait for each other within circular chain. The problem with two robots is solved by Algorithm 4. However, the problem with more than two robots can not be solved merely with the traffic rules. Therefore, a deadlock detection mechanism is used to automatically detect this problem.

The robot which executes one of the *Wait* behavior, sends a so-called *WaitMessage* which contains its ID to every partner at a certain frequency (we use 2 second intervals during our experiments). If its partner is also doing a *Wait* behavior, it will check whether its partner's ID is already contained in this message. Otherwise, its partner will just ignore this message. For the case that its partner's ID is already contained in this message, a chain deadlock is detected and its partner has to plan an alternative path. Otherwise, its partner will add its own ID to this message and forward this message to those robots which this partner is waiting for.

Now we discuss the capability of this method in avoiding deadlocks. Outside of the docking region, the path collision within a certain group of robots executing *FollowWayPoint*, *Avoid*, *Exchange*, or *GoThrough* behavior will be solved with the traffic rules. According the traffic rules, only one robot can execute the *GoThrough* behavior while the other robots have to wait. Inside the docking region, the *Dock*

behavior dominates all other *Active* behaviors. On the assumption that there is no overlap between the docking regions of any stations (Target Isolation), it is impossible for two robots to do the *Dock* behavior in the same region at the same time. Therefore, a cycling of the *Active* behaviors within the same group of robots can not occur. This will ensure that at least one robot in this group will keep moving toward its goal.

All *Active* behaviors except the *Dock* can degrade to one of the *Wait* behavior. In this case the robots can build a cyclic dependency and wait for each other. This problem is solved by the chain deadlock detection and re-planning method as described above. If the robot can always find an alternative path from any position to any station (Alternate Connectivity), at least one robot will be released from the deadlock and move towards its goal.

In summary, our approach is dead-lock free for the problems in which the conditions of Target Isolation and Alternate Connectivity are met.

5.4 Experimental Results

For evaluating our approach we conduct a series of experiments in three different environments at the simulator *Stage*. Figure 5.5 depicts those three environments used to test our approach. The size of the Building_78 map is 33m x 30m, the ASE map is 94m x 82m, and the KNO map is 88m x 43m. On each map we define several loading stations (green rectangle): 4 stations on Building_78, 9 on ASE, and 6 on KNO.

The simulator *Stage* [115] is modified to match our particular demand, for example to illustrate the front area, paths and the name of a behavior. One advantage of the *Stage* is that it allows us to build simulation worlds directly from grid maps, which are generated from real environments in our case.

A comparison between the BBMRCA and the Adaptive Road Map Optimization (ARMO), introduced in Chapter 4, and the decoupled technique based on prioritized planning (PRIOR) from Berg and colleagues [111], will be presented in the following sections. In PRIOR, the path is planned iteratively after a pre-defined priority scheme. When the i th robot plans its path, the paths of the $i - 1, i - 2, \dots, i - k$ robot planned beforehand are considered as dynamic obstacles. In order to guarantee completeness with decoupled planners, it is required that the start and goal locations of each robot are so-called garage configurations, i.e., configurations that are not part of $\mathcal{C}_{free}$ of any other robot.

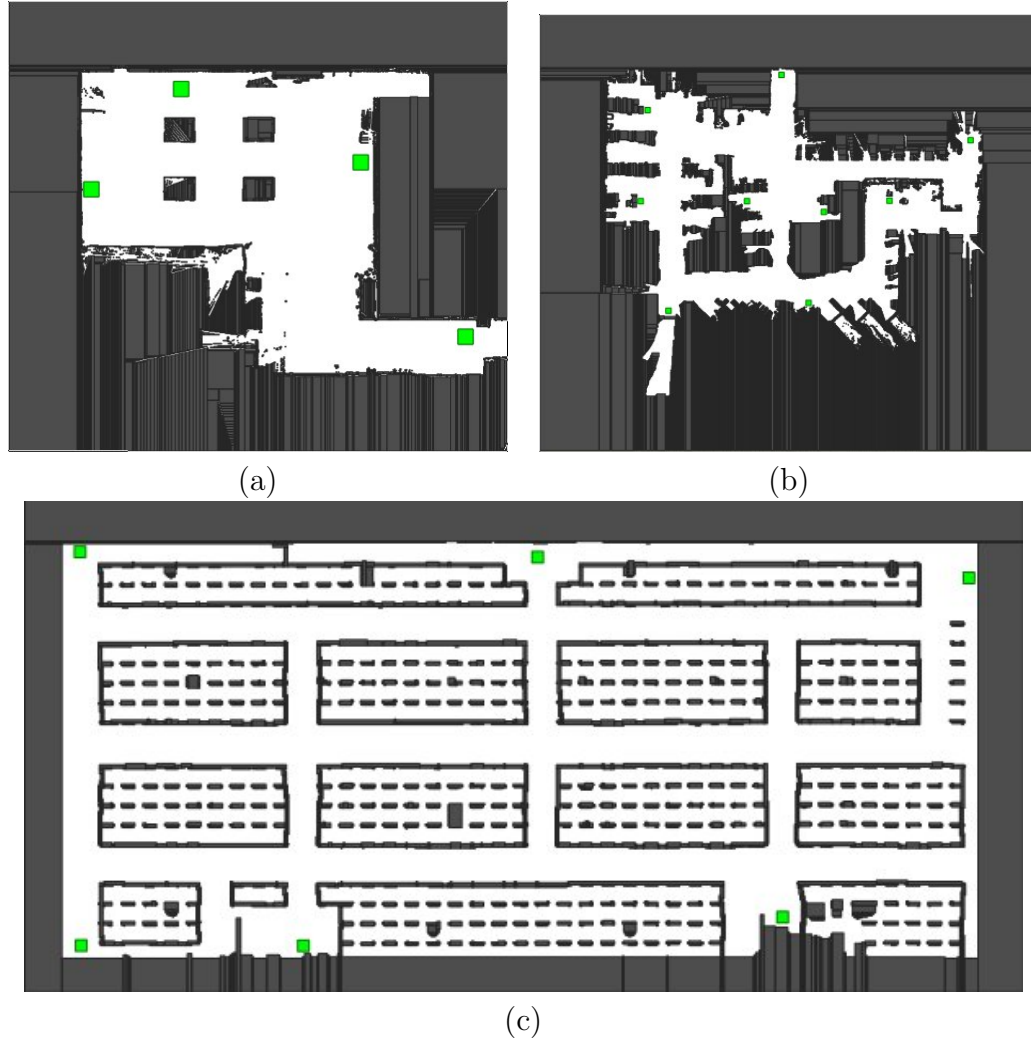


FIGURE 5.5: Snapshot pictures captured from Stage utilized for experiments: (a) the *Building_78* map generated in robot experimental area of University of Freiburg, (b) the *ASE* map generated in a real logistic center. (c) *KNO* map generated in a large distribution center.

Under the same settings, several experiments with the same number and identical sequence of transportation tasks (#Del.) are conducted for each planning method. The maximum allowed velocity of the robot is set to 0.8 m/s for all three methods. The completion times (CTime) is measured in seconds, which reflects mainly the efficiency of the different methods. Additionally, the average velocity over all robots (avg. v) is also measured in meter per second. The average velocity not only provides a different perspective to the efficiency, but also indicates the energy consumption of robots. Normally, move needs more energy than standing; in other words, lower velocity means less energy consumption at the same time period. All experiments are tested with a machine configured with Intel(R) Core i7-3770 (3.4 GHz) processor, 8 GB main memory and Ubuntu 12.04 64-Bit version.

Table 5.1 shows the results of experiments on the map “Building_78”. The free

Building_78				
#R	M	#Del.	avg. v (m/s)	CTime (s)
20	ARMO	144	0.23	1416
	PRIOR	144	0.13	4544
	BBMRCA	144	0.12	2560
40	ARMO	216	0.12	2144
	PRIOR	216	0.09	9144
	BBMRCA	216	0.09	3889

TABLE 5.1: Experiment results of map Building_78.

Map Name ASE				
#R	M	#Del.	avg. v (m/s)	CTime(s)
20	ARMO	196	0.35	2438
	PRIOR	196	0.24	4636
	BBMRCA	196	0.31	2401
50	ARMO	294	0.20	2732
	PRIOR	294	0.15	8229
	BBMRCA	294	0.20	3250
100	ARMO	294	0.11	2826
	PRIOR	294	0.14	10206
	BBMRCA	294	0.16	2685

TABLE 5.2: Experiment results of map ASE.

space of this environment is relatively small, particularly when it is crowded with 20 to 40 robots. Therefore, it is not surprising that ARMO has the best results. However, ARMO requires much more time for computing the optimal road map. Moreover, since the robots can only move on the computed road map and the optimization of such road map is not all the time, another weakness of ARMO is that its reaction time to dynamic changes is not as short as that of the BBMRCA. Furthermore, with almost the same energy consumption, the efficiency of BBMRCA is nearly double of that of PRIOR (44% with 20 robots and 57% with 40 robots). That means, BBMRCA is more energetically efficient than PRIOR.

As is shown in Table 5.2, for the map “ASE”, the performance of BBMRCA is nearly as good as that of ARMO and even better in some cases (with 20 robots and 100 robots). One reason for this is that the road map generated by ARMO contains only orthogonal edges, and in this environment, the orthogonal lines may not be the shortest path between stations. Compared to PRIOR, BBMRCA has a significantly better performance and the results have shown an improvement up to 70%. Additionally, BBMRCA is more scalable by varying the number of the robots.

KNO				
#R	M	#Del.	avg. v (m/s)	CTime(s)
20	ARMO	112	0.40	1746
	PRIOR	112	0.25	4184
	BBMRCA	112	0.36	1902
50	ARMO	196	0.29	1743
	PRIOR	196	0.16	8131
	BBMRCA	196	0.25	2604
100	ARMO	252	0.16	2114
	PRIOR	252	0.14	17977
	BBMRCA	252	0.19	2612

TABLE 5.3: Experiment results of map KNO.

The results of experiments on the map “KNO” are shown in Table 5.3. Although the drivable space of this environment is very limited, the performance of our method is at the same level with ARMO. The average velocities of ARMO and BBMRCA are also nearly identical. Furthermore, by comparison with PRIOR, the efficiency of BBMRCA leads to an improvement up to 85%. And as can be seen in this table, the completion time of PRIOR is raised significantly with increasing number of robots, while the number of robots has hardly any impact on the efficiency of BBMRCA.

5.5 Conclusion

In this chapter we propose a behavior-based multi-robot collision avoidance (BBMRCA) method to solve the MRMP problem in large robot teams. This method is inspired by biological studies on insects and animals which can solve the group navigation problem efficiently. We combine the advantage of the swarm techniques and decoupled multi-robot path planning methods together to achieve an efficient navigation with better quality for large distributed robot systems.

We evaluate our methods with up to 100 robots in a simulation environment and the experimental results suggest that the quality of solutions and the scalability are at the same level with ARMO and significantly better than those of conventional decoupled prioritized planning methods. The main advantage of our approach compared to ARMO is that BBMRCA does not need any pre-processing such as computing the optimal road-map, which requires much time for large and complex environments and is not flexible enough. Our method is as flexible as PRIOR, when an unexpected and unknown obstacle appears. At the same time, compared

to common swarm technologies, our method is more suitable to complicated, structured and crowded environments, such as confined and cluttered logistic centers and manufacturing plants. Furthermore, this method is also implemented and successfully tested on the real robot system within the “KARIS Pro” project (see more detail in Chapter 6 Section 6.3).

Although at present our method is specially designed for the intra-logistic applications, in the future, the behaviors and rules can be relatively easily extended to and adopted in other multi-robot navigation scenarios, for instance, virtual agents in video games.

Chapter 6

The KARIS System

6.1 Introduction

In this chapter, we introduce the KARIS (*Kleinskalige Autonomes Redundantes Intralogistiksystem*) system in more detail. As is mentioned above, the KARIS system is a distributed multi-robot system developed by several companies and research institutes within two joint research projects. The first one “KARIS” is a project of the “Intralogistic Network” in South Germany from 2009 to 2013, and the follow-up project “KARIS Pro (KARIS in der Production)” [83] was funded by a Bundesministerium für Bildung und Forschung (BMBF) grant from 2014 to 2016. The long-term goal of the two projects is to provide a fully autonomous transport system to the flow of materials in the field of intra-logistics and production. The system should work together with humans intuitively and safely, and enable companies to improve the efficiency of their production facilities while reducing the costs.

6.2 The KARIS Project

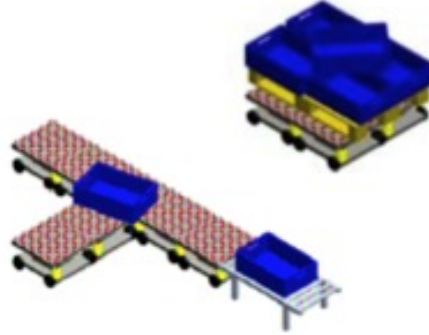
The KARIS project is a joint project from the technology working group of the “Intralogistic Network” in South Germany during 2009-2013. Several companies and research institutes collaborated on the project. Three research groups of our institute have participated in this project. The research group on Computer Networks and Telematics took the responsibility of the communication and network. The research group on Autonomous Intelligent Systems took the responsibility of

the localization. And our research group on Foundations of Artificial Intelligence took the responsibility of the task allocation and multi-robot navigation.

The third prototype of the KARIS mobile robot from this project (see Figure 6.1 (a)) has a size of $50 \times 50 \times 45$ (length x width x height) cm, a payload of 60 kg, and multiple robots can be connected together to build a static conveyor system as is shown in Figure 6.1 (b).



(a)



(b)

FIGURE 6.1: (a) The third prototype of KARIS robot. (b) Inter-robot connections facilitated by a docking mechanism for jointly solving tasks.

The robot is equipped with a holonomic drive system, a conveyor system, a fine docking mechanism and two laser range finders. The holonomic drive system with four Mecanum wheels allows the robot to move in any direction which improves the agility and facilitates the docking behaviors. The conveyor system can automatically loads and unloads crates when docked with a loading station. The fine docking mechanism enables a high precision automatic docking maneuver with the help of four distance measure sensors, either with other robots or with a loading station. Two SICK S300 laser range finders (LRFs) are mounted in two opposing corners to cover the 360 degree field of view for autonomous and safe navigation. The software is developed in C++ and the communication between different software components is based on the Inter Process Communication framework (IPC) [96]. Figure 6.2 depicts an abstract functional software architecture from a viewpoint of most important functions.

For localization, a Monte-Carlo localization method [18] is used. The occupancy grid map [26, 73] is generated with a fast SLAM method [72] from the data collected by once steering a single robot manually through the environment. Furthermore, in the case of using ARMO method a road map based presentation of the environment is used for the multi-robot navigation. The road map is generated from a dynamic occupancy grid map which presents the dynamic changes of the

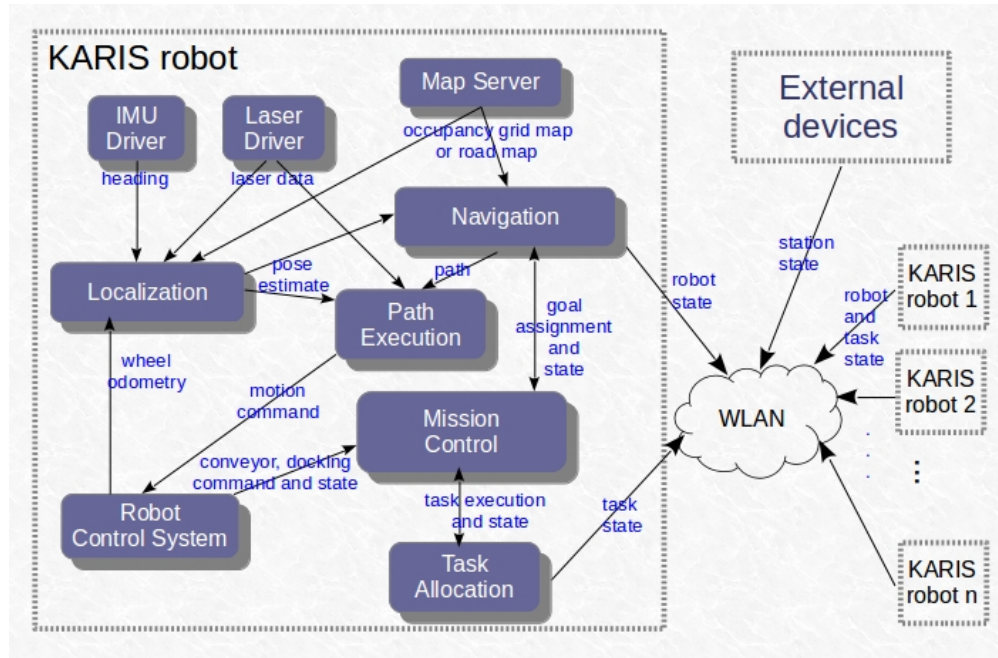


FIGURE 6.2: The abstract functional software architecture.

environment over time (see Chapter 4). For the navigation either the prioritized planning method (PRIOR) from Berg and colleagues [111], or the ARMO method described in Chapter 4 is used. For the task allocation, the DHHT based approach proposed in Chapter 2 is applied.

6.2.1 Application: A Real World Demonstration

In order to show the advantages of the KARIS system against conventional transport systems, we demonstrated the KARIS system at the *Logimat* Fair in Stuttgart 2010. We constructed a real world instantiation of transportation tasks with four KARIS robots (see Figure 6.3 (f)), whose task was to deliver prepared coffees and snack foods to visitors waiting at the delivery stations and to put used cups into the small catering kitchen. The size of the demonstration area was $6m \times 14m$ and there were five delivery stations in total installed in the scenario (see Figure 6.3 (a)). One station next to the kitchen was used to provide catering services and other four stations were used for ordering foods and coffees by visitors. The coffees and foods were put into a small red crate, as is shown in Figure 6.3 (c), thus a particular delivery task was to transport a small crate from one station to one of the other four stations and vice versa.

During this demonstration up to four robots ran continuously for three exhibition days (every day 8 hours) without any interruption. Every day, the robots traveled

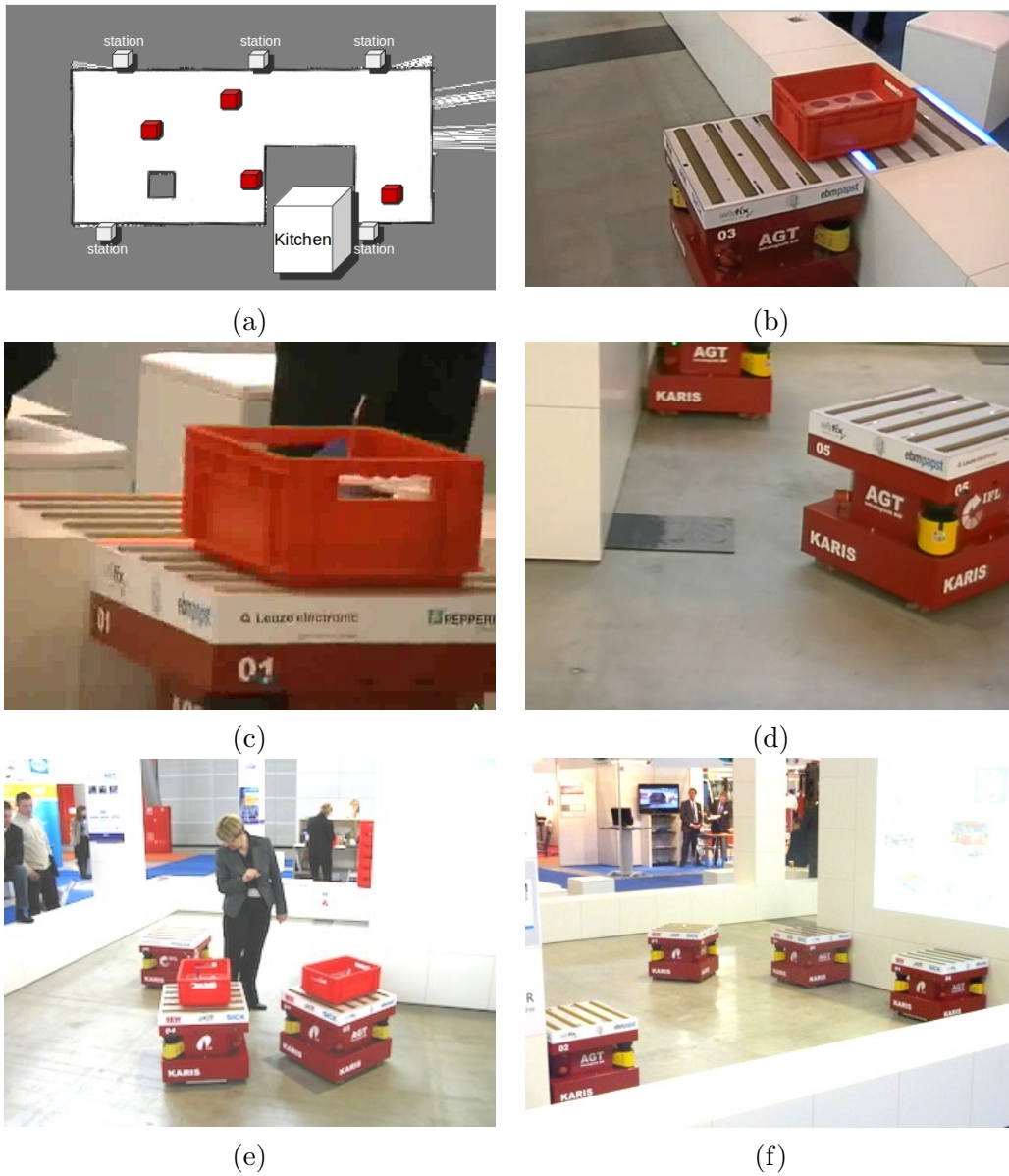


FIGURE 6.3: The demonstration at the *Logimat* Fair. (a) The scenario of the demonstration and the red cube denotes a KARIS robot. (b) A robot is docked with a station and is loading a crate. (c) A red crate used to transport the foods and coffees. (d) A robot is approaching a recharging position. (e) The safe navigation among humans. (f) Four KARIS robots used in this Demonstration.

on average four kilometers and finished on average 180 delivery tasks without collisions or deadlocks. We demonstrated successfully that the KARIS system is able to organize the transportation tasks by itself in human workspace in an efficient and flexible way.

6.3 The KARIS Pro Project

The KARIS Pro project is the follow-up project of KARIS funded by a Bundesministerium für Bildung und Forschung (BMBF) grant during 2014-2016. The plan was to have two KARIS Pro pilot applications in real production environments by 2016. Numerous companies and research institutes from Baden-Württemberg collaborated on the projects (see Figure 6.4), and our institute took the responsibility for localization and navigation.

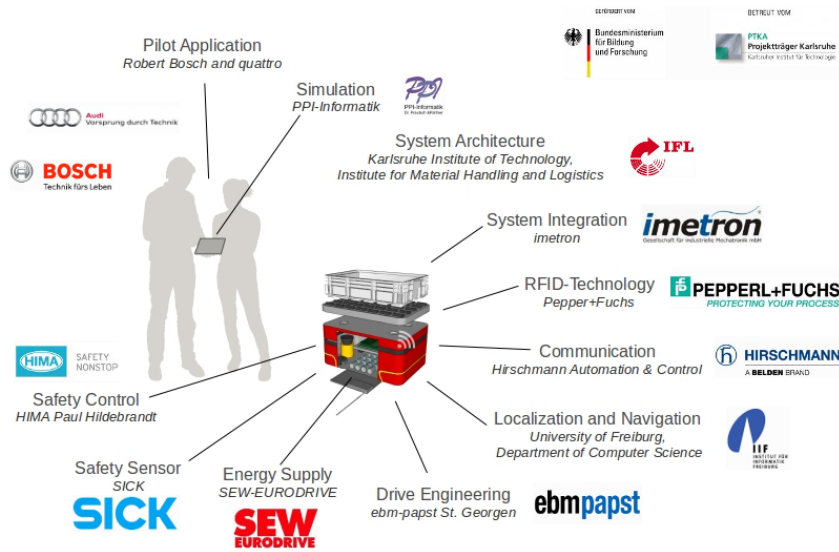


FIGURE 6.4: The multi-company research project KARIS Pro with our institute taking the responsibility for localization and navigation.

In this project, the robot is completely re-designed and produced. Figure 6.5 shows the final version of the robot. The size of the robot is $60 \times 40 \times 40$ (length x width x height) cm, a payload of 120 kg and is equipped with a holonomic drive system, one SICK S300 laser range finder mounted in front of the robot and a exchangeable module for two different transportation tasks. One module is designed for the small load carrier (see Figures 6.5 (a)) and the robot can be automatically docked with a loading station to load or unload the carrier. The other module is designed for a cart (see Figure 6.5 (c)) and the robot can autonomously dock under the cart and move together with the cart. The robot is powered with an on-board power system and the battery can be charged automatically by docking to a contact-less charger station (see Figure 6.5 (b)). The software is based on the Roboter Operating System (ROS) [84] libraries. To deal with the high dynamics in real environments, the dynamic localization method described in Chapter 3 is implemented and applied. To increase efficiency and flexibility, the behavior-based

multi-robot collision avoidance method described in Chapter 5 is applied for the multi-robot navigation. Furthermore, a scan matching based fine position method is implemented in order to realize the autonomous docking function with a loading or charge station. The fine position reaches an accuracy of 1 cm (see Figures 6.5 (a)).

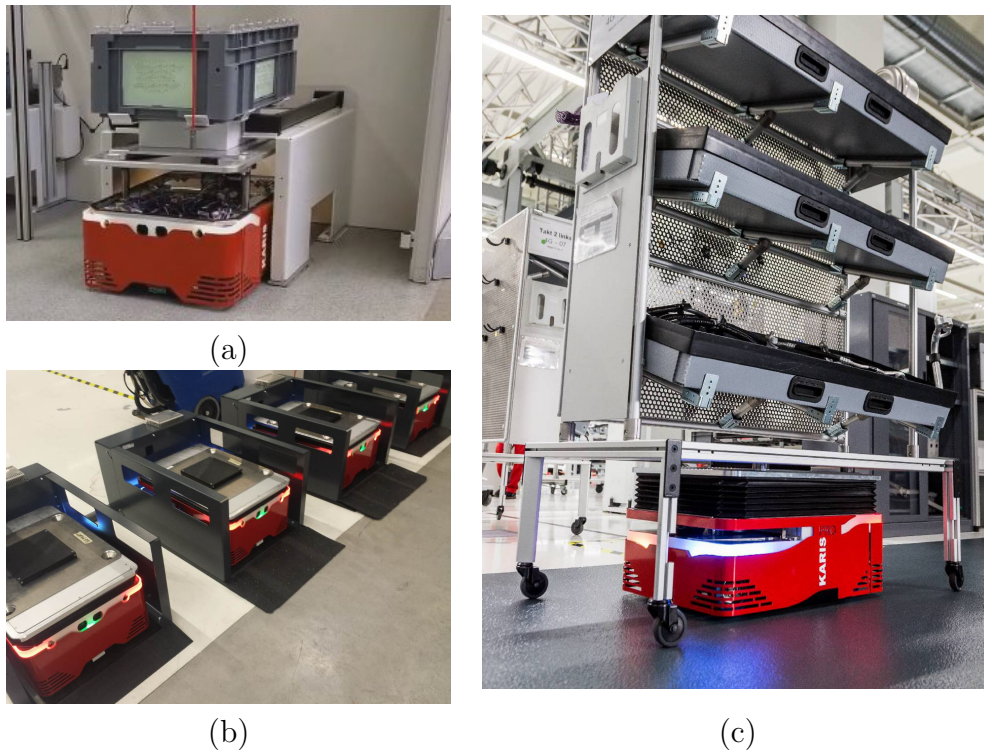


FIGURE 6.5: (a) A robot equipped with a module for small load carrier is docking with a loading station. (b) Robots are charging with contact-less charger stations (c) A robot docks with a cart and moves together with it.

6.3.1 Two Pilot Applications

In this project two pilot applications are successfully completed. The first pilot application was carried out for the Bosch Diesel Systems, Stuttgart, Germany, in the field of intra-logistics. The second pilot application was carried out for the Audi R8 production of the quattro GmbH, Neckarsulm, Germany, in the field of production.

In the first application at the Bosch Diesel Systems, the job is to transport goods in carriers from different production areas to a measuring room and vice versa (see Figure 6.5 (a)). A transport job can be generated manually by entering an order on a tablet by workers or directly from RFID tag mounted on the carrier. All robots are informed about the job via WLAN and negotiate among themselves

by a decentral auction based job management implemented by the Institute for Material Handling and Logistics [44] to decide which robot does which job. Then a robot who takes an order will pick up a carrier and bring it to the destination.

In the second pilot application at the Audi R8 production, the job is to transport carts from the warehouse to the assembly line and vice versa (see Figure 6.5 (c)). The carts containing all the parts needed for a certain production step should be transported to a certain assembly site at the correct time. At the same time the empty carts from assembly sites should be transported back to the warehouse. The transport job is generated manually by entering an order on a tablet by workers. Then, jobs will be allocated to robots in the same way as described in the first application. A robot that takes an order will drive to the cart, dock under it and bring it to the destination.

To ensure that materials could be transported to the right place on schedule safely and fully autonomously, localization and navigation need to overcome many challenges. The difficulty here for the localization is to overcome huge working spaces plus high dynamic influences. For example, the working space in the first application is about 100m x 120m extending over two buildings, and that of the second application is about 80m x 50m. The dynamic influences here include long-term changes caused by rearrangement of shelves or local renovation of plants, as well as short-term changes caused by gathering humans or moving objects such as forklifts or micro goods trams. Our dynamic localization method provided in Chapter 3 works well under such difficult conditions. At the end of the pilot we demonstrate that the pose tracking problem in real environments with high dynamics can be solved with our approach.

The difficulty here for the navigation is to deal with complex traffic situations in real industrial environments. In addition to avoid collisions with each other, the robot must also face other participants in traffic, such as pedestrians, forklifts, micro goods trams and even fast-moving bicycles. Furthermore, in some places our system should follow the traffic rules used by other on-site transport systems. For this purpose, a road map is manually built to fulfill the special requirements of different scenarios. For instance, in the road map an edge can be marked as a one-way street, so that the moving direction of our robot is the same as that of the micro trams and an overtaking of it can be prevented. A scan matching based object detection method can be specified at individual nodes on the road map, so that our robot can yield the right of way to micro trams or on-site running AGVS in the Audi scenario. The path planning of a single robot will take into account these specific features of the road map to find an optimal path which meets the

customers' needs. The path execution is controlled by the behavior-based multi-robot collisions avoidance (BBMRCA) method described in Chapter 5. At the end of the project, our method was successfully tested in different situations and achieved a safe, flexible and efficient multi-robot navigation.

With two pilot applications, it has been shown that the KARIS system is able to deal with different real environments in industrial settings, and can bring benefit to customers by reducing of the processing time and saving human resources. It has also been demonstrated that the KARIS system is very flexible and requires little effort to integrate itself into the running production process. Furthermore, we have also acquired many experience and skills from this project, which will help us to design and build a better system in the future. In summary, it has been demonstrated that such a autonomous multi-robot transport system is a realistic, flexible and efficient solution for material transport tasks in the field of intra-logistics and production.

Chapter 7

Conclusions

In this thesis we focus on three relevant problems involved in designing and building a distributed multi-robot system to solve the materials transport problems in the industrial-like domain. The first important problem is the Multi-Robot Task Allocation (MRTA) problem, namely, to decide which object should be transported by which robot and when it should happen. The MRTA problems associated with the multi-robot navigation problem is $\mathcal{NP}$ -hard, so almost all centralized approaches with the lack of scalability can not be applied to large problems. Therefore, we will not attempt to solve such problems directly or exactly, but try to provide an approximation to improve the overall performance of the system. For this purpose, in Chapter 2 we propose a novel approach called Distributed Heterogeneous Hash Tables (DHHT) to solve the task allocation problem. The assignment of a robot to a task by using of the DHHT is divided into two steps: firstly, a robot is assigned to a station with the DHHT method; then, the robot gets a specific transportation task from this station with the help of a contract net protocol. The robots build a kind of Mobile Content-Addressable Network (MCAN) and in the MCAN all messages are delivered via a so-called geographic routing scheme.

This approach is completely decentralized, meaning that it needs no central server to coordinate the task allocation. Only robots and loading stations carry communication devices and all necessary communication is performed by the robots and the loading stations. Therefore, this approach can be used in many production environments without installing additional hardware infrastructures. Moreover, because of the decentralized structure of the algorithm, the system is robust, fail-tolerant and scalable. Failures of some robots or adding further robots during the runtime barely has influence on the system, and the changing demands of stations can be easily adapted by the system. Besides the theoretical findings for the allocation mechanism, we evaluate our approach both in simulation and on the real

system. The experimental results from the simulation and real world have shown that our approach performs a fair distribution of robots among stations under several different test constraints, and thus achieves high total delivery efficiency of the system.

Another important problem is the localization in highly dynamic environments, namely, to determine the robot position in continuously changing environments. While common approaches are successfully applied in static environments, they may not be well-suited to highly dynamic and complex environments, such as intra-logistics and production. In Chapter 3, we introduce an effective and robust dynamic localization method which separates localization from the mapping process to solve such problems. In localization process, a particle filter coupled with a distance filter and a scan matching method achieves a more robust localization against dynamic obstacles. In mapping process, we introduce an extend sensor model which takes dynamic influences as well as the state transition probability of the map directly into the map update step and leads to a more accurate and robust map update. The empirical evaluation shows that our approach achieves a robust localization in several highly dynamic real-world environments, all including dynamic influences. At the same time the mapping stays consistent with the ground truth.

Besides the task allocation and localization problems, another relevant problem involved in the multi-robot system is the Multi-Robot Motion Planning (MRMP) problem, namely, to find out how a team of autonomous mobile robots can move in the same environment at the same time while avoiding collisions with other robots and obstacles. In general, MRMP problems are also $\mathcal{NP}$ -hard. Therefore, coupled approaches are not suitable for a large team of robots. The most conventional decoupled motion planning approaches for real robot systems, like the prioritized path planning (PRIOR) method, are suffering from the unsatisfactory quality of the solution. That means, the quality of the solution can be fairly low with the huge number of robots (e.g up to 100 robots). So we seek for some new methods to improve the quality of solutions in these cases.

In Chapter 4, we propose a road map based approach called Adaptive Road Map Optimization (ARMO) to deal with multi-robot path planning (MRPP) problem. Our approach generates an optimal road map with consideration of environment constraints and the current demands of stations in a pre-processing phase. The robots then extract individual paths from the road map with a local path planner and cooperatively navigate themselves with a local navigation module. The optimal road map will be recomputed only when changes of environments or stations are detected. Furthermore, the computation time of the generation of an optimal

road map depends mainly on the complexity of the environment rather than on the number of robots. Therefore, our approach is capable for a large robot team (up to 100 robots) in dynamically changing environments. We conduct several experiments in simulation to test our approach under different conditions. The experimental results have shown that the ARMO leads to a more efficient multi-robot navigation than conventional decoupled prioritized path planning methods.

Although the ARMO approach has a significantly better quality than the conventional decoupled methods and can be used on real robot systems, a centralized pre-processing is still needed and some constraints are not flexible enough for arbitrary applications, for instance, the road map must be orthogonal straight lines. Through biological research on insects, ants and birds in nature, it is found that the coordination problem in a large numbers of such animals can be highly efficiently solved with swarm techniques. The key point of a swarm system is that a swarm behavior can be triggered automatically by relatively simple rules followed by individuals. Many swarm systems have already been developed for robotics and video games, for example, formation methods, flocking methods, schooling methods and following methods. But few of them are perfectly fit for applications in which robots must move in a structured and crowded environment. Therefore, in Chapter 5, we propose a behavior-based multi-robot collision avoidance (BBMRCA) method for solving MRMP problem in large robot teams. We combine the advantage of the swarm techniques and decoupled multi-robot path planning methods to achieve better flexibility and quality for the real robot system.

We evaluate the BBMRCA method with up to 100 robots in a simulation and the experimental results indicate that the quality of solutions and the scalability are at the same level as those of ARMO and significantly better than those of conventional decoupled prioritized planning methods. The main advantage of our approach compared to ARMO is that BBMRCA does not need any pre-processing such as computing the optimal road-map, which requires much time for large and complex environments. The BBMRCA method is as flexible as PRIOR, when an unexpected and unknown obstacle appears. At the same time, compared to common swarm technologies, this method is more suitable in complicated, structured and crowded environments, such as confined and cluttered logistic centers and manufacturing plants. Although at present our method is specially designed for the intra-logistic applications, in the future, the behaviors and rules can be relatively easily extended to and adopted in other multi-robot motion planning scenarios, for instance, for virtual agents in video games.

Finally, we introduce a demonstration and two pilot applications within two projects “KARIS” and “KARIS Pro” in Chapter 6. The demonstration was carried out at

the *Logimat* Fair in Stuttgart 2010. We successfully demonstrated the advantages of the system compared with conventional transport systems. The first pilot application was carried out for the Robert Bosch GmbH, Stuttgart, Germany, in the field of intra-logistics. The second pilot application was carried out for the quattro GmbH, Neckarsulm, Germany, in the field of production. With these applications, it has successfully been demonstrated that the KARIS system is a realistic, flexible and efficient solution for the materials transport problems in the industrial-like domain.

In summary, we present methods for a distributed multi-robot system to solve materials transport problems in industrial environments. We propose a fully decentralized task allocation method which can facilitate a fair distribution of robots among the loading stations. We introduce an effective and robust localization method to achieve an accurate localization in highly dynamic real world environments. We propose two decoupled multi-robot motion planning methods which can efficiently navigate a huge group of robots in structured and crowded environments. With the experiments in the simulation and on real robots, it has been demonstrated that this system is able to organize the transportation task by itself in human workspace, and solve the transportation problems in real environments in a more flexible and efficient way.

Bibliography

- [1] Swarm robotics: a review from the swarm engineering perspective. *Swarm Intelligence*, 7(1), 2013.
- [2] Ravindra K. Ahuja, Thomas L. Magnanti, and James B. Orlin. *Network flows: theory, algorithms, and applications*, volume 1. Englewood Cliffs, N.J.: Prentice Hall, 1993.
- [3] Dragomir Anguelov, Rahul Biswas, Daphne Koller, Benson Limketkai, Scott Sanner, and Sebastian Thrun. Learning hierarchical object maps of non-stationary environments with mobile robots. In *In Proc. of the Conf. on Uncertainty in Artificial Intelligence (UAI)*, 2002.
- [4] B. Balaguer, S. Balakirsky, S. Carpin, M. Lewis, and C. Scrapper. US-ARSim: a validated simulator for research in robotics and automation. In *Workshop on “Robot Simulators: Available Software, Scientific Applications, and Future Trends” at IEEE/RSJ*, 2008.
- [5] J. Barraquand and J.-C. Latombe. Robot motion planning: A distributed representation approach. *International journal of robotics research*, 10:628–649, 1991.
- [6] M. Bennewitz, W. Burgard, and S. Thrun. Optimizing schedules for prioritized path planning of multi-robot systems. In *Proc. of the IEEE International Conference on Robotics & Automation (ICRA)*, 2001.
- [7] Dimitri P. Bertsekas. Auction algorithms for network flow problems: A tutorial introduction. *Computational Optimization and Applications*, 1:7–66, 1992.
- [8] R. Biswas, B. Limketkai, S. Sanner, and S. Thrun. Towards object mapping in non-stationary environments with mobile robots. In *Intelligent Robots and Systems, 2002. IEEE/RSJ International Conference on*, volume 1, pages 1014–1019 vol.1, 2002.

-
- [9] C. Blum and D. Merkle. *Swarm Intelligence: Introduction and Applications*. Natural Computing Series. Springer, 2008.
 - [10] Manuele Brambilla, Carlo Pinciroli, Mauro Birattari, and Marco Dorigo. Property-driven design for swarm robotics. In *Proceedings of 11th International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)*, 2012. Accepted for publication.
 - [11] J. Bruno, E. G. Coffman, Jr., and R. Sethi. Scheduling independent tasks to reduce mean finishing time. *Comm. Assoc. Comp. Mach.*, 17(7):382–387, July 1974.
 - [12] Rainer E. Burkard, Eranda Çela, Panos M. Pardalos, and Leonidas S. Pit-soulis. The quadratic assignment problem, 1998.
 - [13] R.E. Burkard, M. Dell’Amico, and S. Martello. *Assignment Problems, Revised Reprint*. Society for Industrial and Applied Mathematics (SIAM, 3600 Market Street, Floor 6, Philadelphia, PA 19104), 2009.
 - [14] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper. USARSim: a robot simulator for research and education. In *Proc. of the 2007 IEEE Intl. Conf. on Robotics and Automation (ICRA)*, pages 1400–1405, Roma, Italy, 2007. Citeseer.
 - [15] H. Choset, , and Burdick J. Sensor-based exploration: The hierarchical generalized voronoi graph. *Int. Journal of Robotics Research*, 19(2), 2000.
 - [16] Xavier Décoret and François X. Sillion. Street Generation for City Modelling. In *Architectural and Urban Ambient Environment*, Nantes France, 2002.
 - [17] Morris H DeGroot. *Optimal statistical decisions*, volume 82. John Wiley & Sons, 2005.
 - [18] F. Dellaert, D. Fox, W. Burgard, and S. Thrun. Monte carlo localization for mobile robots. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, 1998.
 - [19] M. Bernardine Dias and et al. Market-based multirobot coordination: A survey and analysis, 2006.
 - [20] E. W. Dijkstra. A note on two problems in connexion with graphs. *NUMERISCHE MATHEMATIK*, 1(1):269–271, 1959.

- [21] M. Dorigo. *Optimization, Learning and Natural Algorithms*. PhD thesis, Politecnico di Milano, Italy, 1992.
- [22] M. Dorigo, M. Birattari, R. O’Grady, L.M. Gambardella, F. Mondada, D. Floreano, S. Nolfi, T. Baaboura, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A.L. Christensen, A. Decugnière, G.A. Di Caro, F. Ducatelle, E. Ferrante, J. Guzzi, V. Longchamp, S. Magnenat, J. Martinez Gonzalez, N. Mathews, M.A. Montes de Oca, C. Pinciroli, G. Pini, P. Réturnaz, F. Rey, J. Roberts, F. Rochat, V. Sperati, T. Stirling, A. Stranieri, T. Stuetzle, V. Trianni, E. Tuci, A.E. Turgut, and F. Vausard. Swarmanoid, the movie. In *AAAI-11 Video Proceedings*, 2011. Winner of the AAAI-2011 Best AI Video Award.
- [23] M. Dorigo, D. Floreano, L.M. Gambardella, F. Mondada, S. Nolfi, T. Baaboura, M. Birattari, M. Bonani, M. Brambilla, A. Brutschy, D. Burnier, A. Campo, A.L. Christensen, A. Decugnière, G.A. Di Caro, F. Ducatelle, E. Ferrante, A. Förster, J. Guzzi, V. Longchamp, S. Magnenat, J. Martinez Gonzalez, N. Mathews, M.A. Montes de Oca, R. O’Grady, C. Pinciroli, G. Pini, P. Réturnaz, J. Roberts, V. Sperati, T. Stirling, A. Stranieri, T. Stuetzle, V. Trianni, E. Tuci, A.E. Turgut, and F. Vausard. Swarmanoid: a novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics & Automation Magazine*, page in press, 2012.
- [24] M. Dorigo, E. Tuci, F. Mondada, S. Nolfi, J.-L. Deneubourg, D. Floreano, and L. M. Gambardella. The SWARM-BOTS project. *Kunstliche Intelligenz*, 2005, in press.
- [25] Frederick Ducatelle, Gianni A. Di Caro, Alexander Förster, Michael Bonani, Marco Dorigo, Stéphane Magnenat, Francesco Mondada, Rehan O’Grady, Carlo Pinciroli, Philippe Réturnaz, Vito Trianni, and Luca Maria Gambardella. Cooperative navigation in robotic swarms. *Swarm Intelligence*, 8(1):1–33, 2014.
- [26] A. Elfes. Sonar-based real-world mapping and navigation. *Robotics and Automation, IEEE Journal of*, 3(3):249–265, June 1987.
- [27] M. Erdmann, T. Lozano-Pérez, MASSACHUSETTS INST OF TECH CAMBRIDGE ARTIFICIAL INTELLIGENCE LAB., and Massachusetts Institute of Technology. Artificial Intelligence Laboratory. *On Multiple Moving Objects*. AI memo. Massachusetts Institute of Technology, Artificial Intelligence Laboratory, 1986.

- [28] Eliseo Ferrante, Ali Emre Turgut, Cristián Huepe, Alessandro Stranieri, Carlo Pinciroli, and Marco Dorigo. Self-organized flocking with a mobile robot swarm: a novel motion control method. *Adaptive Behaviour*, 20(6):460–477, 2012.
- [29] Paolo Fiorini and Zvi Shiller. Motion planning in dynamic environments using velocity obstacles. *The International Journal of Robotics Research*, 17(7):760–772, 1998.
- [30] Dieter Fox, Wolfram Burgard, and Sebastian Thrun. Markov localization for mobile robots in dynamic environments. *Journal of Artificial Intelligence Research*, 11:391–427, 1999.
- [31] D. Gale. *The Theory of Linear Economic Models*. Economics / mathematics. University of Chicago Press, 1960.
- [32] G. Gallagher, S.S. Srinivasa, J.A. Bagnell, and Dave Ferguson. Gatmo: A generalized approach to tracking movable objects. In *Robotics and Automation, 2009. ICRA '09. IEEE International Conference on*, pages 2043–2048, May 2009.
- [33] M.R. Garey and D.S. Johnson. *Computers and intractability : A guide to the theory of NP-completeness*. W. H. Freeman & Co., New York, NY, USA, 1979.
- [34] Brian P. Gerkey and Maja J. Mataric. Sold!: Auction methods for multirobot coordination, 2002.
- [35] Brian P. Gerkey and Maja J Mataric. A formal analysis and taxonomy of task allocation in multi-robot systems. *The International Journal of Robotics Research*, 23(9):939–954, September 2004.
- [36] Silvia Giordano and Ivan Stojmenovic. Position-based ad hoc routes in ad hoc networks. In Mohammad Ilyas, editor, *The Handbook of Ad Hoc Wireless Networks*, chapter 16. CRC Press, 2003.
- [37] G. Grisetti, R. Kümmerle, C. Stachniss, U. Frese, and C. Hertzberg. Hierarchical optimization on manifolds for online 2d and 3d mapping. In *Proc. of the IEEE Int. Conf. on Robotics and Automation (ICRA)*, Anchorage, AK, USA, May 2010.
- [38] G. Grisetti, C. Stachniss, and W. Burgard. GMapping – OpenSLAM.org. <http://www.openslam.org/gmapping.html>.

- [39] Giorgio Grisetti, Cyrill Stachniss, and Wolfram Burgard. Improved techniques for grid mapping with Rao-Blackwellized particle filters. *IEEE Transactions on Robotics*, 23(1):34–46, 2007.
- [40] Wolfgang Hatzack and Bernhard Nebel. The operational traffic control problem: Computational complexity and solutions. In *A. Cesta and D. Borrajo (eds.), Proceedings of the 6th European Conference on Planning (ECP 2001)*, 2001.
- [41] A.J.W. Hilton. The reconstruction of latin squares with applications to school timetabling and to experimental design. In V.J. Rayward-Smith, editor, *Combinatorial Optimization II*, volume 13 of *Mathematical Programming Studies*, pages 68–77. Springer Berlin Heidelberg, 1980.
- [42] H. Hippenmeyer, K. Furmans, T. Stoll, and F. Schöning. Ein neuartiges Element für zukünftige Materialflusssysteme. *Hebezeuge Fördermittel: Fachzeitschrift für Technische Logistik*, (6), 2009.
- [43] J.E. Hopcroft and J.T. Schwartz. *On the Complexity of Motion Planning for Multiple Independent Objects; Pspace Hardness of the Warehouseman’s Problem*. BiblioBazaar, 2011.
- [44] IFL. Institute for material handling and logistics, 2016. [Online; accessed 21-Dec-2016].
- [45] E.B.I.R.F.S.F. Institute, M.D.I.R.F.R.A.B.F.N.R. Scientifique, and G.T.R.A.F.C.N. de la Recherche Scientifique. *Swarm Intelligence : From Natural to Artificial Systems: From Natural to Artificial Systems*. Oxford University Press, USA, 1999.
- [46] S. Hughes M. Koes J. Wang, M. Lewis and S. Carpin. Validating usarsim for use in hri research. In *Proceedings of the Human Factors and Ergonomics Society 49th Annual Meeting*, pages 457–461, 2005.
- [47] S. Jain, K. R. Fall, and R. K. Patra. Routing in a delay tolerant network. In Raj Yavatkar, Ellen W. Zegura, and Jennifer Rexford, editors, *SIGCOMM*, pages 145–158. ACM, 2004.
- [48] M. Jünger, T.M. Lieblich, D. Naddef, G.L. Nemhauser, W.R. Pulleyblank, G. Reinelt, G. Rinaldi, and L.A. Wolsey. *50 Years of Integer Programming 1958-2008: From the Early Years to the State-of-the-Art*. Springer, 2009.

- [49] M. Kallman and M. Mataric. Motion planning using dynamic roadmaps. In *Proc. of the IEEE Int. Conf. on Robotics & Automation (ICRA)*, volume 5, pages 4399–4404, 2004.
- [50] K Kant and S W Zucker. Toward efficient trajectory planning: The path-velocity decomposition. *Int. J. Rob. Res.*, 5(3):72–89, September 1986.
- [51] D. Karaboga. An idea based on honey bee swarm for numerical optimization. Technical Report TR06, Erciyes University, October 2005.
- [52] D. Karger, E. Lehman, T. Leighton, M. Levine, D. Lewin, and R. Panigrahy. Consistent hashing and random trees: Distributed caching protocols for relieving hot spots on the World Wide Web. In *Proceedings of the Twenty-Ninth Annual ACM Symposium on Theory of Computing*, pages 654–663, El Paso, Texas, 1997.
- [53] B. Karp and H. T. Kung. GPSR: greedy perimeter stateless routing for wireless networks. In *Proceedings of the 6th Annual ACM/IEEE International Conference on Mobile Computing and Networking (MobiCom 2000)*, pages 243–254, 2000.
- [54] L.E. Kavraki, P. Svestka, J.-C. Latombe, and M.H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. *Robotics and Automation, IEEE Transactions on*, 12(4):566–580, August 1996.
- [55] Lydia Kavraki, Petr Svestka, Jean claude Latombe, and Mark Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces. In *IEEE INTERNATIONAL CONFERENCE ON ROBOTICS AND AUTOMATION*, pages 566–580, 1996.
- [56] J. Kennedy and R. Eberhart. Particle swarm optimization. In *Neural Networks, 1995. Proceedings., IEEE International Conference on*, volume 4, pages 1942–1948 vol.4. IEEE, November 1995.
- [57] Serge Kernbach, Eugen Meister, Florian Schlachter, Kristof Jebens, Marc Szymanski, Jens Liedke, Davide Laneri, Lutz Winkler, Thomas Schmickl, Ronald Thenius, Paolo Corradi, and Leonardo Ricotti. Symbiotic robot organisms: Replicator and symbion projects. In *Proceedings of the 8th Workshop on Performance Metrics for Intelligent Systems, PerMIS '08*, pages 62–69, New York, NY, USA, 2008. ACM.
- [58] Soheil Keshmiri and Shahram Payandeh. Multi-robot, dynamic task allocation: a case study. *Intelligent Service Robotics*, 6(3):137–154, 2013.

- [59] O. Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The International Journal of Robotics Research*, 5(1):90–98, Spring 1986.
- [60] Seung kook Yun, Mac Schwager, and Daniela Rus. Coordinating construction of truss structures using distributed equal-mass partitioning. In *in Proc. of the 14th International Symposium on Robotics Research, Lucern*, 2009.
- [61] Alex Kushleyev, Daniel Mellinger, and Vijay Kumar. Towards a swarm of agile micro quadrotors. *Robotics: Science and Systems*, July 2012.
- [62] S. M. LaValle. *Planning Algorithms*. Cambridge University Press, Cambridge, U.K., 2006. Available at <http://planning.cs.uiuc.edu/>.
- [63] Kristina Lerman, Chris Jones, Aram Galstyan, and Maja J. Analysis of dynamic task allocation in multi-robot systems. *International Journal of Robotics Research*, 25:225–242, 2006.
- [64] Han lim Choi, Luc Brunet, Jonathan P. How, and Senior Member. Consensus-based decentralized auctions for robust task allocation. *IEEE Transactions on Robotics*, 2009.
- [65] Lantao Liu and Dylan A. Shell. Optimal market-based multi-robot task allocation via strategic pricing, 2013.
- [66] Tomás Lozano-Pérez and Michael A. Wesley. An algorithm for planning collision-free paths among polyhedral obstacles. *Commun. ACM*, 22(10):560–570, October 1979.
- [67] F. Lu and E. Milios. Globally consistent range scan alignment for environment mapping. *Auton. Robots*, 4(4):333–349, October 1997.
- [68] Ryan Luna and Kostas E. Bekris. Push and swap: Fast cooperative path-finding with completeness guarantees. In Toby Walsh, editor, *IJCAI*, pages 294–300. IJCAI/AAAI, 2011.
- [69] M. Mauve, J. Widmer, and H. Hartenstein. A survey on position-based routing in mobile ad hoc networks. *IEEE Network Magazine*, 15(6):30–39, November 2001.
- [70] Geoffrey J. McLachlan and Thriyambakam Krishnan. *The EM algorithm and extensions*. Wiley series in probability and statistics. Wiley, Hoboken, NJ, 2. ed edition, 2008.

- [71] D. Meyer-Delius, M. Beinhofer, and W. Burgard. Occupancy grid models for robot mapping in changing environments. In *Proc. of the AAAI Conf. on Artificial Intelligence (AAAI)*, Toronto, Canada, July 2012.
- [72] Michael Montemerlo, Sebastian Thrun, Daphne Koller, and Ben Wegbreit. Fastslam: A factored solution to the simultaneous localization and mapping problem. In *In Proceedings of the AAAI National Conference on Artificial Intelligence*, pages 593–598. AAAI, 2002.
- [73] H. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, 1985.
- [74] H.P. Moravec and A. Elfes. High resolution maps from wide angle sonar. In *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, volume 2, pages 116–121, Mar 1985.
- [75] Alejandro R Mosteo and Luis Montano. A survey of multi-robot task allocation. Technical report, Instituto de Investigación en Ingeniería de Aragón (I3A), 2010.
- [76] Edwin Olson. M3RSM: Many-to-many multi-resolution scan matching. In *IEEE International Conference on Robotics and Automation, ICRA 2015, Seattle, WA, USA, 26-30 May, 2015*, pages 5815–5821, 2015.
- [77] N. J. Nilsson P. E. Hart and B. Raphael. A formal basis for the heuristic determination of minimum cost paths. *IEEE Transactions on Systems, Science, and Cybernetics*, SSC-4(2):100–107, 1968.
- [78] L. E. Parker. *Path Planning and Motion Coordination in Multiple Mobile Robot Teams*, chapter Part 13/M, pages 5783–5799. Springer, 2009.
- [79] M. Pfingsthorn. Wireless simulation server – Documentation, 2008. Available at: <http://robotics.jacobs-university.de/VirtualRobots/WSS.pdf>.
- [80] D. T. Pham, A. Ghanbarzadeh, E. Koc, S. Otri, S. Rahim, and M. Zaidi. The bees algorithm, a novel tool for complex optimisation problems. In *Proceedings of the 2nd International Virtual Conference on Intelligent Production Machines and Systems (IPROMS 2006)*, pages 454–459. Elsevier, 2006.

- [81] W.P. Pierskalla and M. Queyranne. *The tri-substitution method for the three-dimensional assignment problem*. Canadian Operational Research Society J., 5:71-81., 1967.
- [82] Giovanni Pini, Arne Brutschy, Marco Frison, Andrea Roli, Marco Dorigo, and Mauro Birattari. Task partitioning in swarms of robots: an adaptive method for strategy selection. *Swarm Intelligence*, 5(3-4):283–304, 2011.
- [83] PTKA. Kleinskaliges autonomes redundantes intralogistik-system in der produktion, 2014. [Online; accessed 21-Dec-2016].
- [84] Morgan Quigley, Ken Conley, Brian P. Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, and Andrew Y. Ng. ROS: an open-source robot operating system. In *ICRA Workshop on Open Source Software*, 2009.
- [85] Lawrence R Rabiner. A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286, 1989.
- [86] S. Ratnasamy, P. Francis, M. Handley, R. Karp, and S. Schenker. A scalable content-addressable network. In *Proceedings of the 2001 SIGCOMM conference*, volume 31, pages 161–172. ACM New York, NY, USA, 2001.
- [87] Michael Rubenstein, Christian Ahler, and Radhika Nagpal. Kilobot: A low cost scalable robot system for collective behaviors. In *ICRA*, pages 3293–3298. IEEE, 2012.
- [88] S. J. Russell and P. Norvig. *Artificial Intelligence: A Modern Approach*. Pearson Education, 2003.
- [89] Y. Nevatia M. Lewis J. Wang S. Carpin, T. Stoyanov. Quantitative assessments of USARSim accuracy. In *Proceedings of PerMIS*, 2006.
- [90] E. Sahin and W.M. Spears. *Swarm Robotics: SAB 2004 International Workshop, Santa Monica, CA, USA, July 17, 2004, Revised Selected Papers*. Lecture Notes in Computer Science / Theoretical Computer Science and General Issues. Springer, 2005.
- [91] C. Schindelhauer and G. Schomaker. Weighted distributed hash tables. In *17th ACM Symposium on Parallelism in Algorithms and Architectures*, pages 218–227, 2005.
- [92] Jacob T Schwartz and Micha Sharir. On the “piano movers” problem. ii. general techniques for computing topological properties of real algebraic manifolds. *Advances in Applied Mathematics*, 4(3):298 – 351, 1983.

- [93] Bruno Siciliano and Oussama Khatib, editors. *Springer Handbook of Robotics*. Springer, 2008.
- [94] R. Siegwart, I.R. Nourbakhsh, and D. Scaramuzza. *Introduction to Autonomous Mobile Robots*. Intelligent Robotics and Autonomous Agents. The Mit Press, 2011.
- [95] David Silver. Cooperative pathfinding. In R. Michael Young and John E. Laird, editors, *AIIDE*, pages 117–122. AAAI Press, 2005.
- [96] Reid Simmons. Inter process communication (IPC). 2004.
- [97] Reid G. Smith. The contract net protocol: High-level communication and control in a distributed problem solver. *IEEE Transactions on Computers*, C.29, 1980.
- [98] Valerio Sperati, Vito Trianni, and Stefano Nolfi. Self-organised path formation in a swarm of robots. *Swarm Intelligence*, 5(2):97–119, 2011.
- [99] Frits C.R. Spieksma. Multi index assignment problems: Complexity, approximation, applications. In Panos M. Pardalos and Leonidas S. Pitsoulis, editors, *Nonlinear Assignment Problems*, volume 7 of *Combinatorial Optimization*, pages 1–12. Springer US, 2000.
- [100] L. Steinberg. *The backboard wiring problem: A placement algorithm*. SIAM Review 3, 37-50, 1961.
- [101] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, pages 149–160, 2001.
- [102] A. Sud, E. Andersen, S. Curtis, M.C. Lin, and D. Manocha. Real-time path planning in dynamic virtual environments using multiagent navigation graphs. *Visualization and Computer Graphics, IEEE Transactions on*, 14(3):526 –538, 2008.
- [103] D. Sun, A. Kleiner, and C. Schindelhauer. Decentralized hash tables for mobile robot teams solving intra-logistics tasks. In *Proc. of the 9th Int. Joint Conf. on Autonomous Agents and Multiagent Systems (AAMAS 2010)*, pages 923–930, Toronto, Canada, 2010.
- [104] D. Sun, A. Kleiner, and T. M. Wendt. Multi-robot range-only slam by active sensor nodes for urban search and rescue. In *Robocup 2008: Robot Soccer World Cup XII*, volume 5399 of *Lecture Notes in Computer Science*, pages 318–330. Springer, 2009.

- [105] Swisslog. Transcar automated guided vehicle, 2014. [Online; accessed 25-Feb-2014].
- [106] S. Thrun. Learning occupancy grids with forward models. In *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, volume 3, pages 1676–1681 vol.3, 2001.
- [107] Sebastian Thrun, Wolfram Burgard, and Dieter Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents)*. The MIT Press, 2005.
- [108] Gian Diego Tipaldi, Daniel Meyer-Delius, and Wolfram Burgard. Lifelong localization in changing environments. *The International Journal of Robotics Research*, 2013.
- [109] Craig Tovey, Michail G. Lagoudakis, Sonal Jain, and Sven Koenig. The generation of bidding rules for auction-based robot coordination. In *In Proceedings of the 3rd International Multi-Robot Systems Workshop, Naval Research Laboratory*. Springer, 2005.
- [110] AliE. Turgut, Hande Çelikkanat, Fatih Gökçe, and Erol Şahin. Self-organized flocking in mobile robot swarms. *Swarm Intelligence*, 2(2-4):97–120, 2008.
- [111] J.P. van den Berg and M.H. Overmars. In *Proc. of the IEEE/RSJ Int. Conf. on Intelligent Robots & Systems (IROS)*, pages 430–435, 2005.
- [112] J.P. van den Berg and M.H. Overmars. Roadmap-based motion planning in dynamic environments. *Robotics, IEEE Transactions on*, 21(5):885–897, 2005.
- [113] Jur Van den Berg, Ming Lin, and Dinesh Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Robotics and Automation, 2008. ICRA 2008. IEEE International Conference on*, pages 1928–1935. IEEE, 2008.
- [114] Rudolph van der Merwe, Arnaud Doucet, Nando de Freitas, and Eric A. Wan. The unscented particle filter. In *Advances in Neural Information Processing Systems 13, Papers from Neural Information Processing Systems (NIPS) 2000, Denver, CO, USA*, pages 584–590, 2000.
- [115] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.
- [116] P. Velagapudi, K. Sycara, and P. Scerri. Decentralized prioritized planning in large multirobot teams. In *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*, pages 4603–4609.

-
- [117] Chieh-Chih Wang, Charles Thorpe, Sebastian Thrun, Martial Hebert, and Hugh Durrant-Whyte. Simultaneous localization, mapping and moving object tracking. *Int. J. Rob. Res.*, 26(9):889–916, September 2007.
 - [118] Ko-Hsin Cindy Wang and Adi Botea. Tractable multi-agent path planning on grid maps. In *Proceedings of the 21st international joint conference on Artificial intelligence*, IJCAI’09, pages 1870–1875, San Francisco, CA, USA, 2009. Morgan Kaufmann Publishers Inc.
 - [119] P.R. Wurman, R. D Andrea, and M. Mountz. Coordinating hundreds of cooperative, autonomous vehicles in warehouses. *AI Magazine*, 29(1):9–20, 2008.
 - [120] Y.Crama, A. Oerlemans, and F.C.R. Spieksma. *Production Planning in Automated Manufacturing*. Springer Verlag, Berlin, 1966.
 - [121] Robert Zlot, Anthony tony Stentz, M. Bernardine Dias, and Scott Thayer. Multi-robot exploration controlled by a market economy. pages 3016–3023, 2002.