Stefan Imlauer, BSc

A Hierarchical Navigation System for Groups of Autonomous Logistics Robots in Industrial Environments

MASTER'S THESIS

to achieve the university degree of

Diplom-Ingenieur

Master's degree programme: Telematics

submitted to

Graz University of Technology

Supervisor

Ass.Prof. Dipl.-Ing. Dr.techn. Gerald Steinbauer

Institute for Software Technology

Graz, March 2016

This document is set in Palatino, compiled with pdfLaTeX2e and `Biber`.

The LaTeX template from Karl Voit is based on KOMA script and can be found online: https://github.com/novoid/LaTeX-KOMA-template

AFFIDAVIT

EIDESSTATTLICHE ERKLÄRUNG

Acknowledgements

I would like to express my gratitude to my supervisor Gerald Steinbauer, who invested lots of hours in interesting discussions on algorithms and methodologies. He provided great support over nearly my entire study, especially during the time of my master thesis.

Additionally I have to thank the company *incubedIT* for funding this thesis. More on I have to thank the entire staff of this great company for supporting me in any questions regarding their system and providing a great possibility to embed my work in a real robotic system. Special thanks goes to Michael Reip, who co-supervised me and supported me throughout my master thesis and contributed with brilliant ideas.

I am also indebted in the staff of the laboratory *Autonomous Intelligent System* group of the *Institute for Software Technology (IST)* and the RoboCup Rescue Team *TEDUSAR*. They supported me anytime with great ideas and lots of coffee. I would like to address special thanks to Clemens Mühlbacher for contributing discussions and lots of implementation details.

Furthermore I have to thank my girlfriend, who has been a great support in many difficult hours throughout my master thesis. Finally I want to thank my family and all of my friends for supporting me throughout my whole life.

Stefan Imlauer
Graz, Austria, March 2016

Abstract

In this work we present a hierarchical navigation system for groups of autonomous logistics robots. The system operates in a cluttered industrial environment with traffic regulating constraints. These traffic rules are intended to ensure efficient and structured navigation for a group of autonomous robots. Furthermore traffic rules are supposed to improve the collaboration between robots and human workers in a shared environment.
The techniques, which are developed in this work, contribute to a hierarchical planning system by allowing an intelligent way of distributed traffic resource allocation. As a basis an expressive environment representation was developed. This representation is an extended graph representing spatial relations as well as traffic regulating constraints for dedicated regions. The resulting graph allows planning algorithms to find efficiently a solution of a planning problem considering different traffic constraints. Moreover we propose high-level planning algorithms which are able to coordinate paths of multiple robots considering such traffic constraints. For this reason we adapted the A* algorithm and developed a heuristic, which is capable to deal with traffic regulating constraints. Multiple robots are using a central reservation timetable for the coordination of the decentralized path planning and common resource allocation. Hence robots are able to optimize their path and arrival time to a goal of a navigation task respecting the paths of the other robots.
Finally the proposed planning system was realized on an industrial robotic system. We provide an extensive evaluation of the proposed algorithms using different performance criteria. This evaluation shows a successful coordination of multiple robots with different traffic region types. Furthermore the evaluation showed a performance gain

compared to an existing implementation especially when considering environments with many traffic regions.

Contents

Abstract vii

1 Introduction 1

1.1 Motivation 2

1.2 Goals and Challenges 3

1.3 Contribution 5

1.4 Outline 6

2 Problem Formulation 7

3 Related Research 11

3.1 Multi-Robot Navigation Systems 11

3.2 Path Planning and Environment Representation 14

3.3 Negotiating Environmental Resources with Respect to Time 17

4 Prerequisites 19

4.1 Robot Operating System 19

4.2 OpenMesh - Half-Edge Data Structure 23

4.3 Allen's Interval Algebra 24

5 Concept 27

5.1 Overview 27

5.2 Environment Representation 29

5.2.1 Polygon-Mesh out of Traffic Regions 30

5.2.2 Road Map Graph 37

5.2.3 Subdivision of Triangles 42

5.3 Region Planner 45

5.3.1 Fastest Path Planner 45

5.3.2 Temporal Heuristic Planner 47
5.3.3 Interval Scheduling 49
5.3.4 Formal Properties of the Region Planner 55
5.3.5 Planner Result 61
5.4 Plan Integration and Execution 64
5.4.1 High-Level Plan Integration 64
5.4.2 Server Communication 65
5.4.3 Central Server Validation 68

6 Implementation Details 69
6.1 Region Parsing 69
6.2 Region Planning 71
6.2.1 Boost R-Tree 72
6.2.2 Server Requests 73
6.2.3 Server Communication and Validation 74
6.3 System Integration and Controlling 75
6.3.1 Planning Procedure 76
6.3.2 Update Planning Graph 79

7 Evaluation 81
7.1 Overview 81
7.2 Evaluation Scenario 82
7.3 Planner Evaluation 83
7.3.1 Worst Case Analysis 86
7.3.2 Subdivision Evaluation 87
7.4 System Evaluation 91
7.5 Use Cases 95

8 Conclusion 99
8.1 Discussion 99
8.2 Future Work 101

Bibliography 103

List of Figures

1.1 Multiple Robots of the *incubedIT* System 2

2.1 Constraints of a Crossing and Right Hand Traffic 9

4.1 Concept of Publisher-Subscriber Model in ROS 21
4.2 Concept of the Service Model in ROS 22
4.3 Concept of the Action Model in ROS 22
4.4 Connectivity Relations in the Half-Edge Data Structure . 24

5.1 Overview of the Navigation System 29
5.2 Exemplary Map with Traffic Regions 30
5.3 Inflation Radius of a Robot Footprint 32
5.4 Obstacle Inflation with Different Methods 32
5.5 Extracted Free Space Polygon 33
5.6 Illustration of the Advanced Region Determination Algorithm 36
5.7 Road Map Generation Techniques out of Triangulations 39
5.8 Triangulation and Connectivity Graph of the Running Example 40
5.9 Illustration of an Oneway Representation 41
5.10 Triangulation Subdivision Techniques 43
5.11 Illustration of the Triangle Inequality for Heuristic Function 47
5.12 Graph Illustration for a THP Invocation 51
5.13 Time Allocation Example Capacity Regions 56
5.14 Visualization of a Non Optimal Plan 59
5.15 Reservation Visualization of a Non Optimal Plan 60
5.16 Planner Result of Empty Single Robot Region 62
5.17 Planner Result of Blocked Single Robot Region 63

5.18 Conceptual Overview of the System Integration 65
5.19 Composition of a Timetable 66
5.20 Concept of the Get Timetable Server Request 66
5.21 Concept of the Reserve Server Request 67
5.22 Concept of the Cancel Server Request 67
5.23 Concept of the Early Entry Server Request 68

6.1 Class Diagram of the RegionParser Module 70
6.2 Class Diagram of the RegionPlanner Module 72
6.3 Sequence Diagram of the RegionPlanning Procedure . . 73
6.4 Class Diagram of the Timetable Manager 75
6.5 Class Diagram of the Controlling Module 77
6.6 Overview of the MoveBase extensions 77
6.7 Sequence Diagram of the Controlling Module 78
6.8 Visualization of Intermediate Goal Planning 80

7.1 Evaluation Scenarios with Increasing Complexity 84
7.2 Planner Evaluation on Three Different Scenarios 85
7.3 Illustration of the Worst Case Analysis 87
7.4 Subdivision Degrees Applied on Level 3 Scenario 89
7.5 Resulting Planner Times for Different Subdivision Degrees 90
7.6 Robots of the *incubedIT* System 92
7.7 System Evaluation on Different Scenarios with Multiple Robots 93
7.8 System Evaluation with Realistic Goal Distribution . . . 94
7.9 Throughput Evaluation of the System 95
7.10 Use Case of Pursuing Robots 97
7.11 Use Case of a simultaneously Goal Execution 98

List of Tables

4.1 Basic Relations of Allen's Interval Algebra 25

7.1 Integration Status of the Traffic Region Types 82
7.2 Tabular Overview of the Evaluation Scenarios 83
7.3 Characteristics of the Subdivided Evaluation Scenarios . 88

1 Introduction

This thesis presents a new approach for a hierarchical navigation system for groups of autonomous logistics robots in industrial environments.
Mobile robot systems has been playing an increasing role in intra logistics scenarios for years. Earlier logistics was dominated by automation technologies for example conveying systems. This basic technologies are necessary for basic functionalities. However in the last decades robots appear in logistics environments and provided more flexibility in comparison to pure automation systems. As a first improvement of flexibility parts of warehouses have been changed from classical conveying systems to line-following robotics. However this is still connected with extensive modifications if a product line changes since the infrastructure for the line-following robots is often static. Although Grossman [1] claimed in his work "Traffic control of multiple robot vehicles" in 1988 that there will be never automated guided vehicles (AGVs) which are navigating fully autonomously in industrial environments comprehensive developments in the robotics fields showed contrary incidents lately. Latest work in the field of robotics takes this to the next level with fully autonomous transport robots in industrial environments.

Within the 4th industrial revolution also referred as *Industry 4.0* many things have already been changed in nowadays industry. Especially in logistics scenarios like warehouses or production areas, robotic systems take over more and more tasks. Industry 4.0 introduces new possibilities to develop intelligent warehouse management systems. In times of short product life cycles and customized products, production lines need to be changed frequently. This is a huge overhead for common production areas and warehouse management system where usually

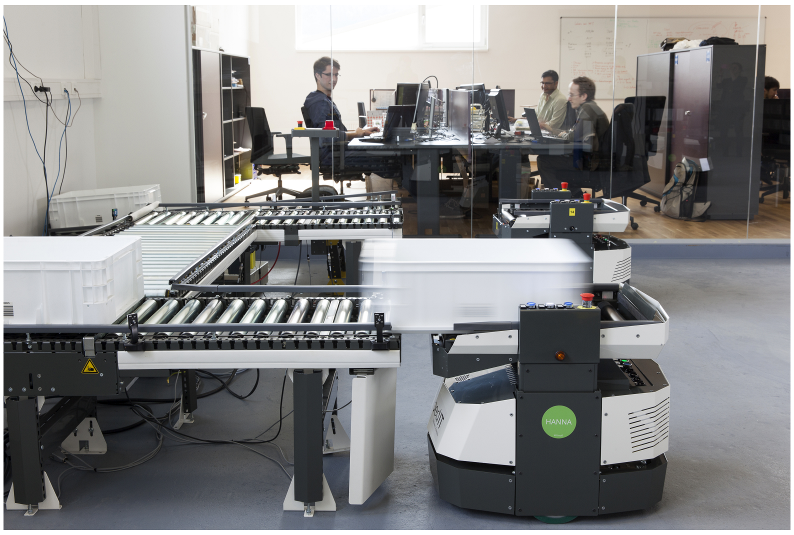


Figure 1.1: Robots of the *incubedIT* system. ©2016 INCUBED IT - AUSTRIA

expensive modification of the environments are necessary. Thus the need of self-navigating AGVs arises in specific situations. However this kind of robotic systems need to fulfill especially safety requirements since in production areas and warehouses the collaboration between robots and human workers is necessary. For this reason also regulations of the environment in the form of traffic rules are imaginable to provide safety for humans and a structured behavior for robots. Moreover the structure of an environment and the throughput of a robotic system generates a demand on traffic rules. This has already been discussed by Cao et al. [2] in 1997.

1.1 Motivation

The company *incubedIT*[1] near the city Graz, Austria developed an autonomous robotic system which faces above described requirements. They are producing a system which embeds an autonomous group of transport robots in intra-logistics scenarios (Figure 1.1). The robots are

[1]`http://www.incubedit.com/`

executing transport tasks from loading stations of conveying systems and allow a highly flexible configuration of production lines. Due to a laser-based navigation method they are not bound to lines on the ground or magnetic stripes to navigate. Thus they also need not to modify the environment if product lines change.
The individual robots are aware of their environment and capable to avoid collisions. Additionally their robots have a certified security laser scanner to guarantee safety, which allows the direct collaboration with human workers without any security fences. In their system every robot is a complete decentralized unit capable of self-localization and self-navigation in a cluttered warehouse environment.
In order to provide a structured behavior within large groups of autonomous robots but also to provide a better acceptance of human workers they introduced traffic regions which represent traffic regulating constraints. These constraints are for example one ways or single robot zones. On the one hand these traffic rules are supposed to provide a predictable behavior of robots and therefore a more comfortable collaboration for human personnel. On the other hand traffic rules are intended to decrease the potential of collisions and deadlocks which leads to a higher robustness and a more structured behavior of groups of robots.
Since every robot is a decentralized unit and basically has no information of the position or tasks of other robots there arises the need of a method to coordinate the group of robots and increase the efficiency of the whole group. Hence a planning layer is missing which is aware of the paths of other robots and able to minimize the duration of a transport task. This work contributes to this system with a hierarchical planning approach among multiple robots which is able to coordinate multiple robots with respect of the traffic region constraints.

1.2 Goals and Challenges

In the following we present the goals and challenges for this system. The previously described system and requirements suggest a system with multiple robots navigating through an environment with traffic

zones. Constraints in this regions limit for example the number of robots in a region. Thus robots have to share these traffic regions which ends in an resource allocation problem. In order to integrate a planning layer able to coordinate multiple robots considering shared traffic resources an adequate environment representation is needed. This representation of the environment has to provide the information of traffic regions (location, size, shape) and their constraints. Additionally a good representation of the spatial relations has to be generated. This is important for the subsequent planning step. Therefore this structure should be an expandable representation which provides the possibilities to integrate information which is presently not efficiently realizable in the currently used grid-based representation.
The challenge of this representation is to provide an efficient structure allowing fast planning. Though this model should be expressive and provide enough information of the environment to generate reasonable plans increasing the efficiency of the overall system.
Moreover an intelligent method of sharing the traffic regions between multiple robots is requested. This method should be capable of an early determination of traffic region capacity overflows and other traffic constraint violations. Therefore the goal is to provide plans which are able to coordinate the usage of the traffic regions and to minimize the arrival of a robot considering other robot paths. This challenges the generation of paths and the scheduling of resource allocations within planning. Furthermore algorithms and good heuristics are required which are able to find efficiently valid plans and provide good estimations of time slots for traffic resource allocations.
A last important goal is to provide a solution not only for academic usage but also applicable in an industrial robotic system. This requires a good handling of available system resources which means that algorithms should be tractable and efficient computing of solutions on a state of the art computation unit should be possible. Additionally existing network resources may not be overloaded when computing a solution for this problem.

1.3 Contribution

This thesis contributes a general formal problem definition of the multi-robot path finding problem with traffic zones as a constrained satisfaction problem. A formulation was developed which describes an optimization problem for finding optimal paths for individual robots executing a navigation task and considering multiple other robots navigating through an environment with traffic regulating constrains. Furthermore a simplification of this problem is shown which makes this problem tractable. In this simplification we formulate an optimization problem which computes an optimal path of an individual robot considering the traffic regulating constrains and the paths of all other robots.
We propose therefore high-level planning algorithms which are able to respect the paths of other robots and provide a solution for the above described reduced optimization problem. Therefore we adapted the A* algorithm and built a heuristic which is able to integrate traffic region constraints into the planning problem.
Furthermore we developed an expressive representation of an environment with areas and traffic regulating constraints attached. This representation allows planning algorithms to find efficiently a solution. The representation is based on a graph constructed out of a triangulated polygon mesh depicting the spatial relations and the traffic region constraints. This representation is automatically built from data given by the industry.
Finally the introduced planning algorithms have been extensively evaluated under different criteria. We showed that the proposed system is capable to produce a performance gain when considering environments with many traffic regions. However even in environments with less traffic regulating constrains the system could keep up with the original system, meaning that the introduced system extensions depicts only a negligible overhead.

1.4 Outline

The remainder of this thesis is organized as follows. In Chapter 2 we present the problem formalization of the given problem as a constrained satisfaction problem. In the following chapters (Chapter 3-4) we related research and present basic prerequisites necessary for building an algorithm to solve the problem and to provide an implementation. In Chapter 5 we present our methodology for solving the hierarchical navigation problem for multiple autonomous logistics robots. Next we describe in Chapter 6 implementation details of the work as well as the integration in an already existing navigation module for autonomous transport robots. Chapter 7 presents the results of an evaluation of the proposed planning system and finally we conclude our findings and discuss possible improvements and future work in Chapter 8.

2 Problem Formulation

This work deals with a multi-robot logistic scenario with a set of n autonomous robots $\mathcal{R} := \{R_1, R_2, \dots, R_n\}$. We define a convex polygon $\mathbb{P}$ as a list of l points $\mathbb{P} := (P_1, P_2, \dots, P_l), P_i \in \mathbb{R}^2, 1 \leq i \leq l$. For the inclusion relation we define the operator $\overline{\in}$ in Equation 2.1.

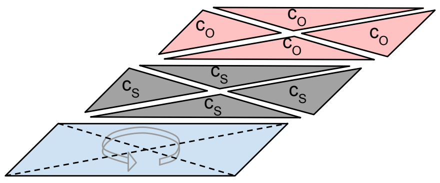
$$\overline{\in} : \mathbb{R}^2 \times \mathbb{P} \rightarrow \begin{cases} 1, \, if \text{ the point lies within the area restricted by } \mathbb{P} \\ 0, \, else \end{cases} \tag{2.1}$$

A robot is moving in an environment $\mathcal{E}$. $\mathcal{E}$ is defined as $\mathcal{E} := \{x \in \mathbb{R}^2 \mid x \overline{\in} \mathbb{P}\}$. Every $R_i \in \mathcal{R}$ is a decentralized unit navigating simultaneously in $\mathcal{E}$. We define a time t where $t \in \mathbb{R}^+$. The position of the robot at time t is defined through the function $\mathbf{pos}\colon \mathcal{R} \times t \rightarrow \mathbb{R}^2$. We define the driving direction vector with the function $\mathbf{dir}\colon \mathcal{R} \times t \rightarrow \mathbb{R}^2$. The length of the vector represents the velocity of the robots. Robots are navigating between a set of k static loading stations $\mathcal{L} := \{L_1, L_2, \dots, L_k\}, L_i \in \mathcal{E}, 1 \leq i \leq k$. A set of navigation tasks $\mathcal{T}$ has to be executed by the multi-robot system. A single task $T \in \mathcal{T}$ is a tuple of $\langle s, g \rangle$ and is assigned to exactly one $R_i \in \mathcal{R}$. Where $s \in \mathcal{L}$ and $g \in \mathcal{L}$ determines the fixed start and goal station of a task. In this work it is assumed that the low-level path planning and path execution is already solved. This thesis concentrates on a region-based traffic planning problem in cluttered environment. Acting on plans with specific traffic rules and regions is supposed to provide a behavior with little potential for collisions and higher efficiency.

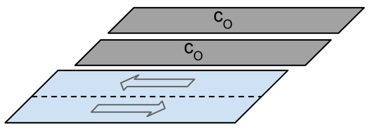
In order to introduce traffic regions within the given environment a set of m areas $\mathcal{A} := \{A_1, A_2, \dots, A_m\}$ is defined. $A_i \in \mathcal{A}$ is defined as a tuple of $A_i := \langle \mathbb{P}_i, \mathcal{C}_i \rangle$. $\mathcal{C}_i \subseteq \mathcal{C}$ denotes a set of possible constraints with $\mathcal{C} := \{c_N, c_S, c_F, c_O, c_D, c_C, c_R, c_V\}$.

In the following the constraints of $\mathcal{C}$ are defined:

- **N-Robots** c_N
  N robots are permitted in an area at the same time. N_{A_i} is defined as the maximum number of allowed robots:
  $$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) \leq N_{A_i} \tag{2.2}$$
- **Single Robot** c_S
  Only one robot is permitted in an area at the same time:
  $$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) \leq 1 \tag{2.3}$$
- **Forbidden** c_F
  No robot is allowed to traverse the area:
  $$\forall t \in \mathbb{R}^+ . \left(\sum_{r \in \mathcal{R}} \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \right) = 0 \tag{2.4}$$
- **One Way** c_O
  This region is only traversable in a specific direction: The direction of this zone is defined by $\mathbf{dir}_{A_i}$.
  $$\forall t \in \mathbb{R}^+, r \in R \,.\, \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \rightarrow \mathbf{dir}(r,t) = \mathbf{dir}_{A_i} \tag{2.5}$$
- **Dynamic One Way** c_D
  The first robot which enters the region, determines the driving direction for this region. No other robot in this area is allowed to drive in the opposite direction:
  $$\begin{split}\forall t \in \mathbb{R}^+ \,.\, \nexists r, r' \in R \,.\, \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \wedge \mathbf{pos}(r',t) \overline{\in} \mathbb{P}_i \wedge \\ \mathbf{dir}(r,t) \neq \mathbf{dir}(r',t) \wedge r \neq r'\end{split} \tag{2.6}$$
- **Velocity** c_V
  The maximal allowed velocity of a region is restricted to $\mathbf{vel}_{A_i}$:
  $$\forall t \in \mathbb{R}^+, r \in R \,.\, \mathbf{pos}(r,t) \overline{\in} \mathbb{P}_i \rightarrow \|\mathbf{dir}(r,t)\| \leq \mathbf{vel}_{A_i} \tag{2.7}$$



(a) Decomposition of the crossing constraint



(b) Decomposition of the right hand traffic constraint

Figure 2.1: Illustration of the constraint decomposition. (c_S) denotes the single robot constraints, (c_O) shows the one way constraints.

- **Crossing** t_C
  A crossing consists of eight areas. Four areas have c_S constraints and four areas do have c_O constraints. A polygon with c_S constraint has to be congruent with a polygon with c_O constraint. The combination of constraints is visualized in Figure 2.1 (a). The c_S constraints allow only one robot in the segment of a crossing. The c_O constraints permit a robot to drive counter clock-wise through a crossing similar to the concept of a roundabout.
- **Right Hand Traffic** c_R
  Robots in this region are forced to drive on the right hand side. This constraint is built with two c_O constraints. The areas corresponding to the constraints have to be adjoined and oppositely directed (Figure 2.1 (b)).

The set of constraints imposed by $\mathcal{A}$ is defined as $\mathbb{C} = \bigcup_{A_i \in \mathcal{A}} \mathcal{C}_i$. A path of a single robot in this domain is defined as a continuous mapping $\pi^{(R_i)}\colon [t_{R_i}, t_{R_i} + \Delta t_{R_i}] \to \mathcal{E}$, referring the book of Choset [3, chap. 3]. The start of the path is furthermore restricted to $\pi^{(R_i)}(t_{R_i}) = s_{R_i}$ and the goal to $\pi^{(R_i)}(t_{R_i} + \Delta t_{R_i}) = g_{R_i}$. The velocity along this path is computed in Equation 2.8. The velocity constraint c_V has to follow the computed velocity.

$$\forall t \in \mathbb{R}^+ \ . \ \mathbf{dir}(R_i, t) = \frac{d\pi^{(R_i)}(t)}{dt} = \begin{bmatrix} \frac{d\pi_x^{(R_i)}(t)}{dt} \\ \frac{d\pi_y^{(R_i)}(t)}{dt} \end{bmatrix} \tag{2.8}$$

The optimization problem results in finding the set of paths Π which minimize the maximal end time $t_{R_i} + \Delta t_{R_i}$ for all paths $\pi^{(R_i)} \in \Pi$ subjected to $\mathbb{C}$ (Equation 2.9).

$$\Pi^* = \min_{\Pi} \left\{ \max_{\pi^{(R_i)} \in \Pi} (t_{R_i} + \Delta t_{R_i}) \right\} s.\ t.\ \mathbb{C} \tag{2.9}$$

The optimal solution for this problem requires the complete information of every task assigned to a robot from the beginning until the end time of the system. Since this information is simply not available we are not able to generate a global optimal solution for the entire robot system. Therefore we define in Equation 2.10 a new planning problem which minimizes the end time of a specific robot subjected to $\mathbb{C}$ and a given set of paths Π from other robots.

$$\pi^{(R_i)*} = \min_{\pi^{(R_i)}} \left\{ (t_{R_i} + \Delta t_{R_i}) \right\} s.\ t.\ \mathbb{C}, \Pi \tag{2.10}$$

The given set Π can be mapped without loss of generality to a set of N tuples $\mathcal{I} = \{I_1, I_2, \ldots, I_n\}, 1 \leq n \leq N$ which represents the traversal of a robot through an area. $I_i \in \mathcal{I}$ denotes the tuple $I_i := \langle t_{s_i}, t_{e_i}, r_i, p_i \rangle$: $t_{s_i}, t_{e_i} \in \mathbb{R}^+, r_i \in \mathcal{R}, p_i \in \mathbb{P}$. This information formulates an additional constraint c_I:

$$\forall I_i \in \mathcal{I} \ . \ \forall t \in [t_{s_i}, t_{e_i}] \ . \ \mathbf{pos}(r_i, t) \overline{\in} p_i \tag{2.11}$$

This constraint offers the possibility to compute an optimal path for a single robot for a given configuration $\mathcal{I}$.

3 Related Research

In this chapter different current research topics and works in literature related to this thesis are discussed. This chapter is composed of four main parts. The first part describes similar multi-robot navigation systems with different fields of application. Then interesting approaches of path planning and navigation systems in mostly single robot scenarios are discussed. Section 3.2 considers also literature dealing with environment representation for path planning. The last section presents some research work on temporal planning and resource allocation.

3.1 Multi-Robot Navigation Systems

Common approaches within multi-robot navigation systems are dealing with decentralized and decoupled path planning methods. Hence, there are no or less shared resources and no centralized mechanisms in order to strengthen scalability. These concepts follow mostly the model of the distributed robot architecture (DRA) [4, chap. 40.2.3].

Kleiner et al. [5] describes a large system of robots operating in logistics centers and manufacturing scenarios. This paper proposes an approach for an efficient decoupled algorithm for multi-robot navigation. Their concept is based on an adaptive road map built out of a grid map. The grid map is produced in advance by an appropriate simultaneous localization and mapping (SLAM) method. To recognize dynamic changes of the environment like pallets, boxes or other obstacles, the grid map is updated dynamically with information of all robots. Therefore every robot reports inconsistencies to the server. Inconsistencies are identified with the help of an extended occupancy

grid map. The authors combined a conventional grid map with an hidden Markov model (HMM) that represents the belief of observation changes. Additionally on the server exists an dynamic occupancy grid map which tracks the reported inconsistencies with an discrete Bayes filter. If the percentage of changed cells in this dynamic grid map exceeds a threshold they publish a new grid map and initiate the road map generation. Out of the dynamic grid map they generate a Voronoi graph which serves as basis for the connectivity network. In this connectivity network stations, locations and crossings are represented as graph nodes and edges serve as connection between the nodes. With a linear programming method they configure the ideal routing of delivery tasks. This takes into account capacities on crossings and lanes as well as flow directions on edges. As a result a road map is configured where packages can be routed with minimal travel costs. Single robots are able to extract shortest paths for their corresponding task with their local planner module.
In this paper the approach is similar to the approach used in this thesis. They also use a centralized computed connectivity graph, which deals as road map for individual robots. However our graph uses a weighted directed graph to represent properties and constraints of traffic regions, but deals not with an optimal configuration in the road map generation step.

The work of Kleiner, Nebel, et al. [6] focuses again on a multi-robot navigation problem in logistics scenarios, but uses a more improved approach to avoid collisions in large robot teams. They use a behavior-based multi-robot collision avoidance method based on decentralized path planning combined with swarm technologies. The path planning part in this method is necessary to handle the complex and cluttered environment of logistics centers. Every robot plans its individual path based on traditional path planning methods, like A*, in a grid map. If a robot comes into contact with another robot, both apply simple behaviors following specific traffic rules. These traffic rules are desired to handle crossings, congestions, docking procedures and more. Inspired by swarm intelligence these behaviors are coordinated only by communicating with direct neighbors of a robot (if they enter a specific area in front of the robot). Hence communication is kept

low and broadcasting is avoided. In the paper they showed that this approach outperforms other multi-robot navigation systems especially if the teams become larger. Another advantage they found is that there is no preprocessing of the environment, for example road maps or graphs, needed.
Although this concept is highly scalable it leads eventually to a reactive approach. Hence it is perfectly able to avoid and negotiate collisions locally but does not consider to handle resources globally, in order to make a likely collision free path available.

Wang and Premvuti [7] introduced in their work a traffic control system for multiple autonomous robots. Robots are able to travel through a discrete traffic network of passages (narrow directed corridors), intersections and terminals. All of these traffic segments have finite capacities. Their work is proposed to be used in scenarios where self-regulated traffic systems are necessary, such as automated guided vehicles on manufacturing floors or automatic roadway/railway systems. This system is again designed as a fully decentralized navigation framework. Thus there is no central computation unit, no shared memory and no ground support which works as an arbiter. Robots are operating in a two-dimensional discrete space consisting of traffic segments which is represented as a static directed graph. Similar to the work of [6] this approach uses only minor intercommunication between robots. Communication is only triggered if they leave or enter traffic segments. Then robots are executing so called "Basic Operating Primitives", which are used to compete for a traffic resource, deadlock detection and deadlock resolution. This is achieved with a *1-out-of-N* methodology.
This paper describes a similar use case to the problem this thesis focuses on. However the work of Wang and Premvuti is again based on local behaviors. The system emerges to a self organizing traffic system, but does not provide possibilities to negotiate time based resources.

The following paper presents a "Constraint-based multi-robot path planning" method [8]. Ryan showed in his work how to solve the problem of planning collision free paths for multiple robots as a constrained satisfaction problem (CSP). In contrast to the work shown previously, this method works as a centralized planning system. The author de-

scribes a planner to coordinate robots with a composite graph that represents all robots. Since this graph grows combinatorially with the number of robots, a method to decompose the graph into subgraphs has been shown. Therefore they exploit structural information of the road map to build subgraphs like cliques, halls, rings and more. This subgraphs constrain the movements of robots. Planning within this subgraphs is encoded as CSP.
Our system does not intend to do centralized planning, but we are able to formulate the planning problem of this work as an constraint satisfaction problem. In our formulation we constrain regions with traffic rules. Furthermore the method of decomposing the graph into subproblems is for some degree similar to the method we handle explicit traffic regions as standalone planning problems.

3.2 Path Planning and Environment Representation

In literature there exists various different path planning techniques using different environment representations. Path planning is still of highly interest in research, in particular in game and robotic research. Especially in this thesis we are focusing on planning techniques that allow hierarchical path planning in abstract environment representations.

A hierarchical path planning method for multi-size agents is described by Harabor and Botea [9]. In their work they developed a new path planning algorithm called "Hierarchical Annotated A*". Normally path finding methods assume simplifications like homogeneous agents and homogeneous environments. These simplifications do not fit if robots have different sizes or are operating in heterogeneous environments (different terrains). The approach in this paper uses a clearance-based abstraction of grid maps. Clearance values are considering obstacles and different terrains. The generated information is annotated to the grid map. Their planning method is based on the classical A* algorithm, but adds robot size and terrain capabilities to the search query,

which produces optimal paths considering the agent's size and capabilities. As this method is not tractable for larger problems they introduced a cluster-based subdivision of the grid map with squared clusters. The discretized adjacent regions are connected with single shortest paths between two clusters. This represents an abstract search graph. The planning problem could now be solved hierarchically with much smaller search spaces.
The method of hierarchical planning is in principle the same we use in our navigation system. The major difference is the environmental representation. Here we use a directed graph generated out of a polygonal representation of the environment. Additionally our graph is annotated with region information. After planning on this graph the search space on lower-level grid map based planners is reduced. On this smaller planning domain motion plans are planned and executed.

Similar problems are solved in the article of Mitchell and Papadimitriou [10]. This work discussed already in 1991 an alternative way of finding shortest paths through a weighted planar subdivision. Finding shortest paths in planes of disjoint persistent polygon obstacles is typically solved by visibility techniques. In this work a method has been proposed that generalizes the shortest path problem with obstacles to a weighted region problem. In this case a plane is subdivided into polygonal regions with an associated weight. The goal is to minimize the total costs according to an Euclidean metric. This method could represent problems like navigating with an autonomous vehicle through an environment with different terrains. Terrains are represented as polygonal patches in a map. In comparison to most path planning techniques in literature Mitchell and Papadimitriou are not using a grid map and heuristics to solve this planning problem, but compute paths guaranteed to be optimal by using the continuous Dijkstra technique. They model their problem domain into a polygonal mesh consisting of only triangle faces. This representation is stored in a data structure which allows fast determination of neighborhood relations, for example a quad-edge data structure. A path with minimal costs is computed in their approach on the basis of Snell's Law of Reflection, which provides a local optimality property analogous to optical models.

The problem this paper deals with has some parallels to this thesis. They consider planning in environments with different terrains represented as simple weighted polygons. In our approach terrains are different traffic regions. The way of subdividing and storing our traffic map is mainly inspired by this paper. The following works gives also some insight in how subdividing could benefit abstract path planning.

The paper of Demyen and Buro [11] deals with a triangulation-based environment representation and corresponding methods of path finding. The authors found a method to exploit the advantages of triangulated planes to do efficient path planning. In their work they proposed an algorithm that finds shortest paths in a polygonal channel consisting of a combination of triangle faces. This channel is computed with their so called "Triangulated A*" which gives a series of adjacent triangles from the start position to the goal position. In order to find an optimal path within this channel they modified the "Funnel Algorithm" [12] to deal with agents of different size. Hwang et al. [13] discusses also methods of handling structured environment with triangulated meshes. The especially found that a subdivision of large triangles in free space improves the quality of paths. Furthermore they optimized the positions of mesh-vertices according to obstacle curvatures.
The abstraction and planning in polygonal environment, in our case a map with traffic regions and static obstacles, is one of our main concerns in this thesis. Therefore we could benefit a lot of the idea of using a Delaunay triangulation as environment representation. Furthermore the proposal of a reduced subsequent search domain through a polygonal channel has been continued. Also the idea of subdividing triangles to provide better paths has been picked up in this thesis. A further step in this direction is made in the work of Kallmann [14]. They built a graph out of a triangulated plane by using the midpoints of triangle edges as graph nodes and dismissed wall edges. In our work we handled the graph construction very similar but edges are also constrained by specific traffic regions.

The subsequent work presents a robust navigation system for indoor robotic systems in cluttered environment [15]. In their research they focus on office like environments: unmodified and shared with people.

In order to deal with these challenges the authors see an importance of reliable sensing and representing the environment. In their paper they describe a full navigation stack that handles all necessary steps for a robust indoor robot platform. This stack includes sensing and representing three-dimensional obstacles, mapping and localization in this environment and path planning. The latter works as a two level planning approach. The first level describes a high-level planning instance, called global planner, which computes paths from an arbitrary start to goal with a high frequency. This is usually accomplished with an A* algorithm. However this level ignores dynamics and kinematics of a robot. This is considered in the second level. Hereby command velocities are computed with the "Dynamic Window Approach" (DWA) introduced by Fox et al. [16].
This paper is of high interest for this thesis, because our planning methods are supposed to work on top of the global planner. We provide a reduced search domain and intermediate waypoints forcing the global planner to execute the abstract optimal plan.

3.3 Negotiating Environmental Resources with Respect to Time

A topic of high interest when dealing with multi-robot systems is the management of task and resource allocation. Although task and order distribution is assumed to be solved in this work, we had to deal with the resource allocation problem. In this work resources are traffic regions which have to be shared with multiple robots. In literature there are many concepts dealing with scheduling and temporal planning in quite different research fields. In the following we present two works that handle problems in multi-robot scenarios and are highly related to this thesis.

The work of Alami et al. [17] describes a way to cope with the cooperation of large robot-teams through incremental plan-merging. They introduce a concept called "Plan-Merging Paradigm" where decentralized plans are coordinated with the help of exchanging information of

their current status. In their paradigm a single robot creates his own egoistic plan but has to perform plan merging operations on it. The operations validates it in the multi-robot context. These operations include communication with other robots or using a shared resource for coordination. With this concept they are able to cope with negotiating resources.
This concept plays an important role in our traffic region scenario. Here we want to coordinate the use of different traffic regions with multiple robots. We are using this idea to coordinate the decentralized planned paths with a centralized shared timetable.

Chatila et al. [18] describes a system enabling a robot to plan its action considering temporal constraints. The authors present a three level hierarchical architecture. The first level is a temporal planning instance providing a sequence of partially ordered tasks with temporal constraints. The second level is a fast refinement step, splitting the tasks into sequences of actions. The final "execution" layer takes care of acting and sensing in the environment. They developed a temporal planner which is able to reason on symbolic and numeric temporal relations. Therefore they represented the planning domain as an indexed time table. This is a two-dimensional array, where a column holds a discrete time point and a row a logical assertion. The temporal constraints are manged via a temporal relation manager. The resulting plan is given through a sequence of ordered tasks with additional time constraints, like duration, starting and end dates.
This work has inspired the representation of temporal intervals of allocated traffic regions in this thesis. We use also a timetable to manage and reason about temporal availability of traffic resources.

4 Prerequisites

In this chapter some basic prerequisites for this work are presented. This includes theoretical parts as well as software frameworks and necessary data structures. The remainder is organized as follows. The first part deals with the robot operating system. Then a library with an implemented half-edge data structure is explained and finally a part of Allen's interval algebra is discussed.

4.1 Robot Operating System

In Quigley et al. [19] the robot operating system (ROS) is introduced. ROS is an open source robotic framework[1]. This framework is no operating system in the traditional sense. It provides a structured communication layer above the host operating system. ROS is designed for heterogeneous multi-computer architectures connected in a peer-to-peer topology. For example for a system where multiple onboard machines are cooperating with offboard computers. Therefore no central server is needed neither onboard nor offboard. Hence, unnecessary traffic over a wireless link can be avoided. ROS supports multiple different programming languages and is completely free and open source. These two aspects are mainly responsible for the huge amount of available tools and packages containing from low-level communication to drivers, as well as perception and reasoning many software modules.

In the following paragraph the main modules of ROS are described. In ROS a system consists of nodes. A Node in this system is a process

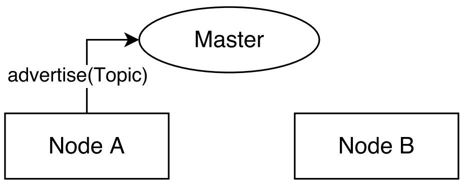
[1]http://www.ros.org/

which performs computation. In a system many nodes are cooperating to solve a task or problem. Nodes are communicating via messages. A message is a small data structure defined with the interface definition language (IDL). The inter-process communication takes place on so called topics. A topic is a named bus on which nodes can exchange information. However nodes need not to be aware of which node they are communicating with, instead they are publishing and subscribing to the topic (Figure 4.1). This kind of broadcasting information has lots of advantages. Nevertheless sometimes a synchronized communication between nodes is necessary. This concept is realized with a service. A service is a pair of well defined messages defining a request and a reply message. The ROS services act analogous to web services. The procedure of a communication based on a service is illustrated in Figure 4.2. In order to manage the nodes and the communication there exists a master. It is responsible to register all nodes as well as to keep track of subscribed and advertised topics and registered services. Once a node has registered itself on the master, the node is visible to all others and communicates peer-to-peer. The master additionally provides a parameter server. This is a shared dictionary on which nodes can share and retrieve parameters at runtime.

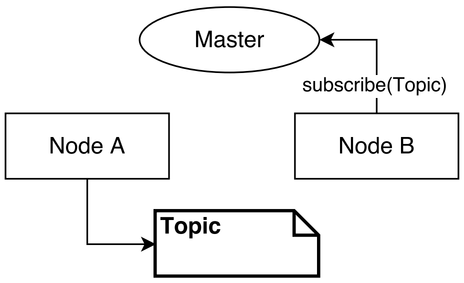
A final module which is brought to ROS by the `actionlib` library provides a possibility to get feedback of a long lasting task and even cancel this task. This is realized similar to the service concept. A node has to provide the action server which handles the computation. This server is requested by another node which implements the action client. This node has the possibility to get periodically feedback from the action server and can even cancel the task. The corresponding concept is shown in Figure 4.3.

An important software package of the ROS community is the *Move-Base*[2]. Marder-Eppstein et al. [15] introduced in their work "The Office Marathon" a software component which allows robust indoor navigation without any modification of the environment. The software component is based on an action server which takes a goal with real world coordinates and combines a path planner (*GlobalPlanner*) and

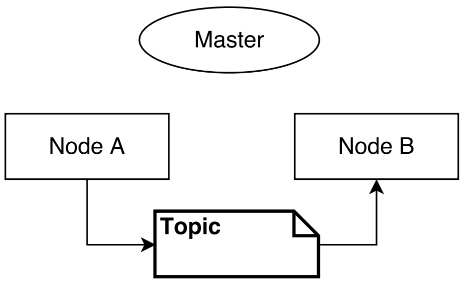
[2]`http://wiki.ros.org/move_base`



(a) Node A advertises a topic to the master

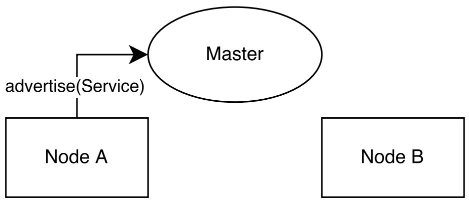


(b) Node A publishes messages to the topic. Node B subscribes to the topic.

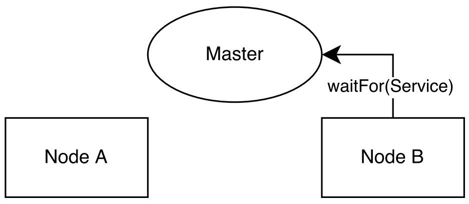


(c) Node B receives messages from the Topic provided by Node A

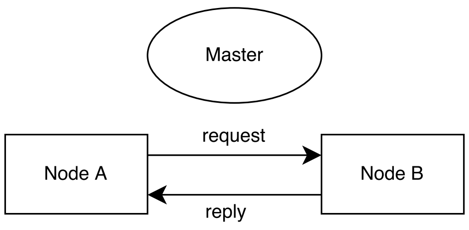
Figure 4.1: Concept of the topic-based publisher-subscriber model in ROS. Adapted from `http://wiki.ros.org/`



(a) Registration of the service on the ROS master



(b) Node B looks up the service on ROS master



(c) Synchronized communication between Node A and B

Figure 4.2: Concept of the service model in ROS. Figures adapted from `http://wiki.ros.org/`

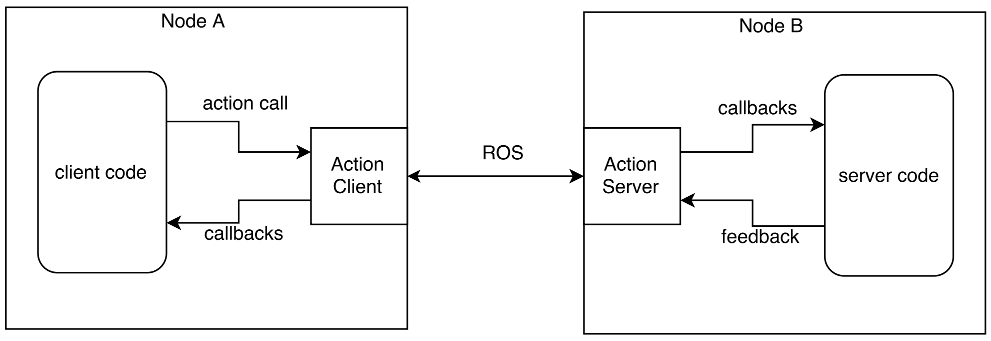


Figure 4.3: Illustration of the action-server and action-client communication. Adapted from `http://wiki.ros.org/`

a planner which computes valid command velocities(*LocalPlanner*). Hence this package provides a complete low-level navigation stack.

4.2 OpenMesh - Half-Edge Data Structure

With OpenMesh[3] Bischoff et al. [20] present an efficient implementation of a half-edge data structure. OpenMesh is a generic and efficient polygon mesh data structure, which is not restricted to triangles, but can handle arbitrary polygon types. Polygon meshes are very important in the field of computer graphics. In this field the representation of arbitrary complex three-dimensional shapes is usually handled with polygon meshes.
A polygonal mesh consist of different elements. Typically a mesh includes faces, edges, vertices and topological relations between them. In order to operate with these elements efficiently, an intelligent data structure is necessary. Relating to these meshes there are existing two kinds of representations: face-based and edge-based data structures. The former stores for each face a pointer to every vertex and neighboring faces. However face-based data structures are usually not able to deal with polygons of different valence. Additionally the one-ring neighborhood is not efficiently computable.
In edge-based data structures pointers from an edge to its adjacent vertices and neighboring edges are stored. This enables the edge-based data structures to handle polygons with different valence.

The underlying data structure in the OpenMesh library is a special case of an edge-based data structure. In this library a so called half-edge data structure is used. This data structure splits every edge in two halves. Therefore the data structure is expanded by the following pointers: every edge points to a vertex, a face and its opposite half-edge (Figure 4.4). A half-edge data structure provides in contrast to a face-based data structure following advantages. Edges, vertices and faces are represented explicitly. This provides the ability to attach any data to those elements. In a mesh one can now easily mix faces with

[3]http://www.openmesh.org/

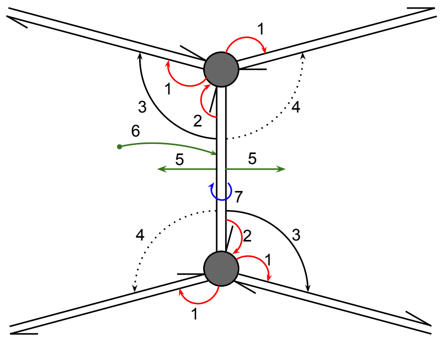


Figure 4.4: This figure shows the connectivity relation in the half-edge data structure. (1) points from a vertex to an outgoing half-edge; (2) from the half-edge to the target vertex; (3) from a half-edge to next half-edge and (4) to the previous half-edge; (5) to half-edge related faces; (6) from a face to one half-edge; (7) to the opposite half-edge

different count of vertices. The one-ring neighborhood of a face can be accessed easily through so called circulators in constant time.

4.3 Allen's Interval Algebra

In the work of Allen [21] a formalism is presented to represent temporal reasoning problems. In more detail Allen introduced temporal intervals for temporal representation. A interval is denoted as an ordered pair of two time points with the first point t_s less than the second t_e. Therefore well defined relationships between intervals exist. Allen's Interval Algebra (AIA) is based on thirteen basic relations (Table 4.1).

The basic relations includes six pairs (during, meet, start, end, before, after) which have a direct converse and one additional relation (equal) which is its own converse. Where the basic relations describe relations between definite intervals, general relations describe interval relations between indefinite intervals. The $2^{13} = 8192$ general relations can be formulated by the combinations of the basic relations. For further reading please see [21].

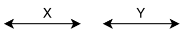
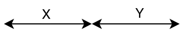
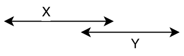
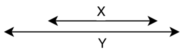
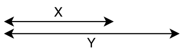
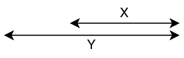
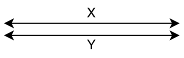
Relation	**Symbol**	**Illustration**	**Endpoint Relation**
X before Y	b		$X_s < Y_s, X_s < Y_e$
Y after X	b'		$X_e < Y_s, X_e < Y_e$
X meets Y	m		$X_s < Y_s, X_s < Y_e$
Y is-met-by X	m'		$X_e = Y_s, X_e < Y_e$
X overlaps Y	o		$X_s < Y_s, X_s < Y_e$
Y is-overlapped-by X	o'		$X_e > Y_s, X_e < Y_e$
X during Y	d		$X_s > Y_s, X_s < Y_e$
Y includes X	d'		$X_e > Y_s, X_e < Y_e$
X starts Y	s		$X_s = Y_s, X_s < Y_e$
Y is-started-by X	s'		$X_e > Y_s, X_e < Y_e$
X finishes Y	f		$X_s > Y_s, X_s < Y_e$
Y is-finished-by X	f'		$X_e > Y_s, X_e = Y_e$
X equals Y	e		$X_s = Y_s, X_s < Y_e$
			$X_e > Y_s, X_e = Y_e$

Table 4.1: Basic relations of Allen's Interval Algebra

5 Concept

In the subsequent sections of this chapter the concept of the developed hierarchical navigation system is presented. This concept is intended to solve the problem of planning with temporally available traffic resources, called traffic regions, in a multi robot logistics scenario.
At the beginning a short overview of the individual parts of this system is given. Then we provide a more detailed view on the representation of the environment. Next planning on this representation is presented and finally the integration and controlling in an existing multi-robot system is presented.

5.1 Overview

The concept of our system relies on a hierarchical planning structure integrated in decentralized units. This planning structure uses a central generated static graph as well as a shared timetable to coordinate the path of all units. The shared timetable is also under central administration. Figure 5.1 gives a schematic overview of our system integrated in an existing multi-robot navigation system. The existing navigation system is based on the work of Marder-Eppstein et al. [15]. This system already includes for each individual robot a complete navigation stack, including a path planning module (*Global Planner*) as well as a *Local Planner* responsible for path following. Every robot operates on a grid map representing the environment. A central server performs tasks like order and goal distribution to the individual robots. Additionally an user can interact with the system via a graphical user interface (GUI) provided by the server. Within this GUI a user can add and modify traffic regions in the environment.

In this work we introduce extensions to this navigation system to enhance the ability of multi-robot coordination. A first important step is the generation of a meaningful representation of the environment. This module is called *Region Parser*. It produces out of the map and the traffic regions a graph constituting spatial relations as well as traffic region properties. The *Region Planner* exploits information annotated to the graph and provides a high-level plan for a single robot. This plan specifies waypoints, restricts the grid map to a planning window which reduces the planning domain for lower level planners and determines a schedule representing the planed entry times along the waypoints. Therefore this planner is capable to determine time intervals on which the robot would need specific traffic regions. Based on these time intervals reservations on the central timetable can be requested. The last step considers controlling and executing the high-level plan. This step includes maintaining the robot's schedule to guarantee the functionality of the system and deal with unpredictable events. Unpredictable events have to be communicated with the server, for example high-level plans canceled or early entries requested. Here interaction with the central *Timetable* and the corresponding *Timetable Manager* plays an important role. Furthermore the lower level planners are forced to follow the waypoints and to consider the waiting times of the high-level plan.

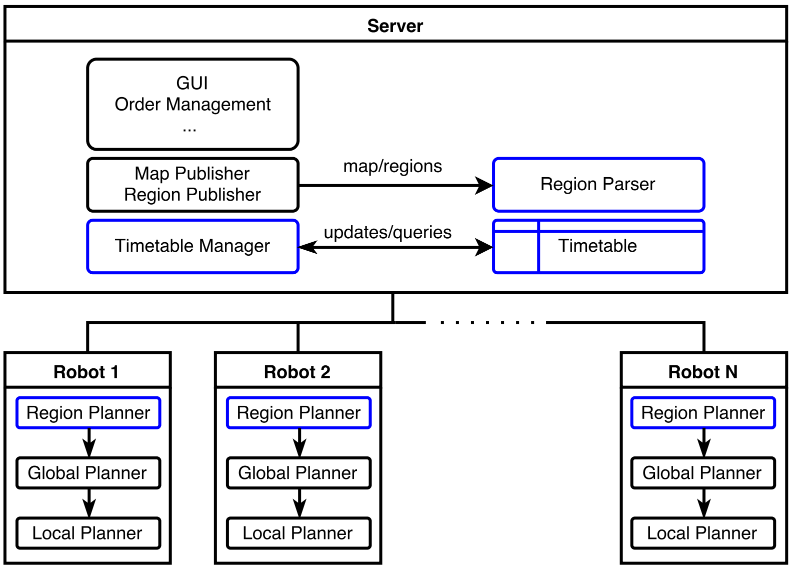


Figure 5.1: Schematic overview of the navigation system. (blue) denotes the novel parts integrated into the existing system (black)

5.2 Environment Representation

In the following algorithms and methodologies for creating an environment representation fitting to our needs are presented. In other research the environment representation problem is often solved by grid map based solutions with four or eight neighbors. The grid cells have obstacle related costs attached. However using grid maps for this problem has some major drawbacks. First to represent the environment well, extremely fine grids are necessary. Additionally grid maps suffer from discretization problems and digitization bias [10]. The representation of environmental costs is also limited within grid maps.
The goal of the presented approach is a graph, capable to represent the traffic region map and to avoid above described problems. A traffic region map comprises multiple polygons which have traffic rules attached. These constrained polygons are furthermore referred as traffic regions. The resulting graph of the environment representation deals as a road map representing paths through free space and the traffic

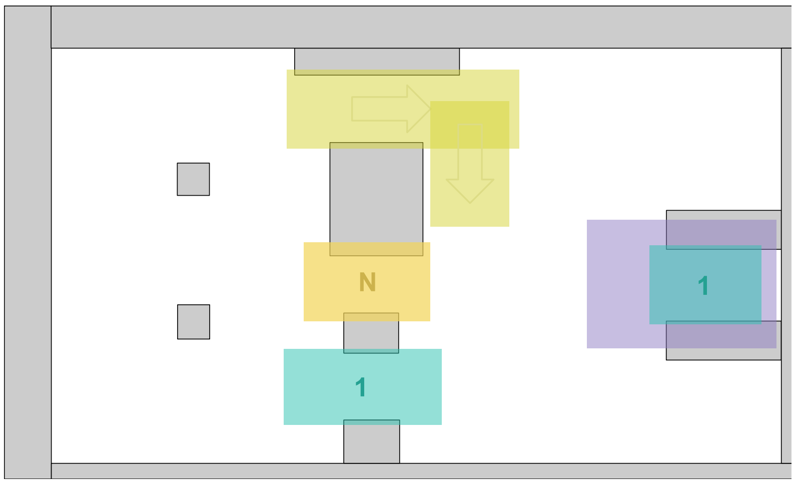


Figure 5.2: Illustration of an exemplary map with traffic regions. This map serves as running example throughout this chapter. The map compromises: oneway traffic regions (yellow); a n-robot region (orange); a single robot region (blue); a simple traffic region (magenta); obstacles (grey); free space (white)

regions. This road map can be computed offline. Updating this graph is only necessary if the shape or the traffic constraints of a traffic region is changed.
Figure 5.2 shows an exemplary map with different region types arranged likewise to a typical real-world problem. This constructed map will deal as running example throughout this chapter.

5.2.1 Polygon-Mesh out of Traffic Regions

Traffic regions are represented in this work as polygons with specific traffic regulating constraints. Robot teams operate in maps composed of these traffic regions. Considering these circumstances a representation is needed where the structure of polygons is not lost and planning is fast and efficient.

Typical path planning methods are dealing with the shortest-path problem with obstacles. In this problem the environment is only modeled as free space and obstacles. Here motion planning can be in principle divided into two categories: sampling-based motion planning and combinatorial motion planning. The former is usually used if the explicit construction of obstacles should be avoided because of high complexity. These methods use a sampling based approach to determine the configuration space. The latter looks for paths trough the configuration space without approximations. In contrast to the former planning techniques the latter are exact algorithms. The combinatorial motion planning algorithms typically operate on road maps. For road map generation there exist several different approaches. Some are generated from cell decompositions of the environment, others directly from environment. So there also exists a wide variety of different algorithms to perform cell decomposition. They are also applicable for different use cases. Some provide advantages for different dimensions, some provide smoother transitions in road maps. [22, chap. 5-6]

In the traffic region scenario a generalization of above described planning and road map techniques is needed. This scenario requires a constitution where polygons are not automatically depicted as obstacles, but have additional properties. Hence, a representation that ideally combines free space, obstacles and traffic regions is needed.

Free Space Determination

In order to get a sequence of traversable traffic regions it is necessary to determine the free space exactly. Furthermore the representation of free space and traffic regions has to be equal. Therefore the following two-step procedure has been used: (1) inflate all obstacles, (2) cut-out all inflated obstacles and traffic regions from the map. In the first step every polygon is expanded with the inflation radius of a robot. The inflation radius is defined through the circumscribed radius of a robot footprint (Figure 5.3). This guarantees that a robot with a fixed geometry will not collide with any obstacle in the environment and that computed paths are traversable by the robot. The abstraction of inflating obstacles is in this work made with a "squared" miter approach (Figure 5.4). In this approach all convex edge joins are approximated

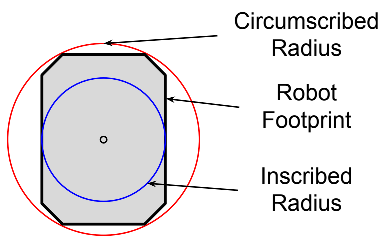


Figure 5.3: Representation of the circumscribed and inscribed radii of a robot footprint. The circumscribed radius is used as inflation radius. Adapted from [15].

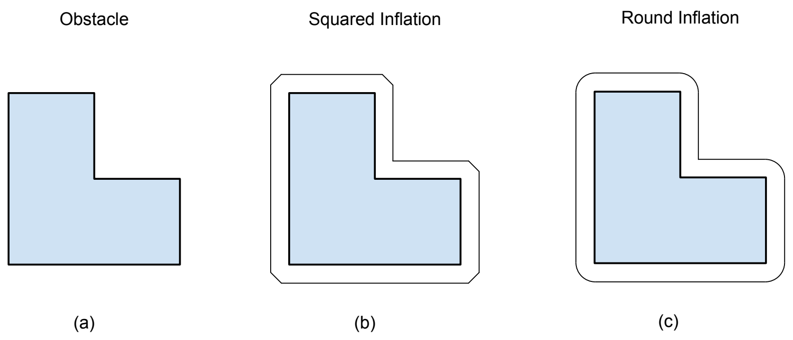


Figure 5.4: Illustration of two obstacle inflation methods. (a) obstacle to be inflated; (b) obstacle inflated with "squared" approximations of all convex edge joins; (c) obstacle inflation with rounded convex edge joins

in the way shown in this figure. Other inflation techniques use a round miter or exact miter representation. The former is disadvantageous for later triangulation since an arc creates many triangles. The latter is critically if the object to be inflated consists of acute angles. The squared miter approach provides a solid and good approximation of edge joins and has been chosen for our free space representation.
The second step creates one or more polygons, which are most likely concave or include multiple holes. These polygons depict the free space in the environment. Figure 5.5 presents the structure of a possible free space polygon with holes.

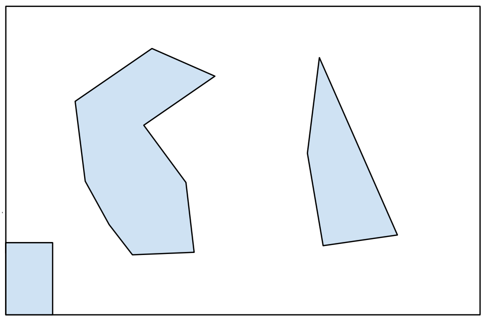


Figure 5.5: Illustration of an exemplary free space polygon (white) with holes, resulting from obstacles (blue) cut-out.

Advanced Region Determination

The traffic region map consists of arbitrary polygons. Every traffic region is associated with a specific region type that constrains the actions of robots. Since positioning of regions in the map is not constrained, regions may also overlap each other. For this reason we consider the overlaps of two or more regions as new regions with the union of properties and constraints. In the following we propose Algorithm 1 which performs this advanced region determination. This algorithm iterates over a list of valid traffic regions. This list comprises all traversable traffic region types. Hence every not traversable traffic region, for example a forbidden area, is declared as invalid traffic region and is excluded. The resulting valid traffic region list includes all traversable and therefore valid traffic regions. In the first step of an iteration the first valid traffic region from the list is reduced by any overlapping obstacle or invalid traffic region. In the inner loop the current region is intersected with every not yet investigated traffic region. If the result of the intersection is empty, the region does not overlap with any other traffic region and the next iteration starts. If the result of the intersection is not empty the region is investigated in more detail. First the difference of the current region *region_i* and the next region in the list *region_j* is computed. If the difference is empty *region_i* is fully enclosed by *region_j*. Thus the traffic constraints of *region_i* are extended with the constraints of *region_j*. If the difference is not empty, the polygon of *region_i* has to be updated with the resulting polygon of the difference. This test is repeated with the difference of $region\_j \setminus region\_i$. If none of the tests results in a fully enclosed region the result of the intersection is added as new region with constraints of *region_i* and *region_j*. In Figure 5.6 this procedure is illustrated with an example.

Triangulation of Polygons

Triangulating the polygonal environment is one of the essential points of our traffic region representation. This cell decomposition method generates for all parts of the environment the same geometric shape and provides the benefits of representing the map using triangles. We use constrained Delaunay triangulation (CDT) to decompose our polygons. The Delaunay triangulation maximizes the minimum angle of all angles in the triangulation. A CDT provides the ability to respect defined constraints, for example predefined edges which have to appear in the triangulation. However this constraints could lead to the loss of the Delaunay condition.[23]
The decomposition of polygons into triangles has two major advantages. A possible concave polygon is reduced to a set of convex polygons. Thus, every edge has complete visibility of all other edges in this polygon (triangle). This means that any straight line between any combination of two edges of this polygon is completely enclosed by the polygon. Secondly a set of triangles brings up a simple but expressive possibility to build a road map. These advantages are exploited in Section 5.2.2.

Representing Polygons in a Data Structure

Due to previous steps the triangulated free space and traffic regions share the same geometric shape. Furthermore the triangles form a polygon mesh. Obstacles result in holes in this mesh. Thus, this mesh is able to combine every polygon type we discussed so far.

Algorithm 1: Advanced Region Determination

```
Data: $valid\_regions$ ... list of valid map regions
Data: $obstacles$ ... list of obstacle polygons
Result: $planner\_regions$ ... list of identified regions for planning
begin
    $new\_regions \longleftarrow \{\}$
    for $i \leftarrow 0$ to $valid\_regions.size()$ do
        $region\_i \longleftarrow valid\_regions[i] \setminus obstacles$

        for $j \leftarrow i + 1$ to $valid\_regions.size()$ do
            $region\_j \longleftarrow valid\_regions[j]$
            $intersection \longleftarrow region\_i \cap region\_j$

            if $intersection \neq \{\}$ then
                $region\_i\_cut \longleftarrow region\_i \setminus region\_j$
                $region\_j\_cut \longleftarrow region\_j \setminus region\_i$

                if $region\_i\_cut \neq \{\}$ then
                    /* polygon of $region\_i$ is updated with
                    $region\_i\_cut$                                  */
                    $region\_i \longleftarrow region\_i\_cut$
                else
                    $region\_i.updateProperties(region\_j\_cut)$
                    $fully\_enclosed \longleftarrow true$
                end
                if $region\_j\_cut \neq \{\}$ then
                    /* polygon of $region\_j$ is updated with
                    $region\_j\_cut$                                  */
                    $region\_j \longleftarrow region\_j\_cut$
                else
                    $region\_j.updateProperties(region\_i\_cut)$
                    $fully\_enclosed \longleftarrow true$
                end
                if $fully\_enclosed = false$ then
                    $new\_regions.add(Properties(region\_i), Properties(region\_j),$

                                    $intersection)$
                end
            end
        end
        $valid\_regions.append(new\_regions)$
    end
    $planner\_regions \longleftarrow valid\_regions$
end
```

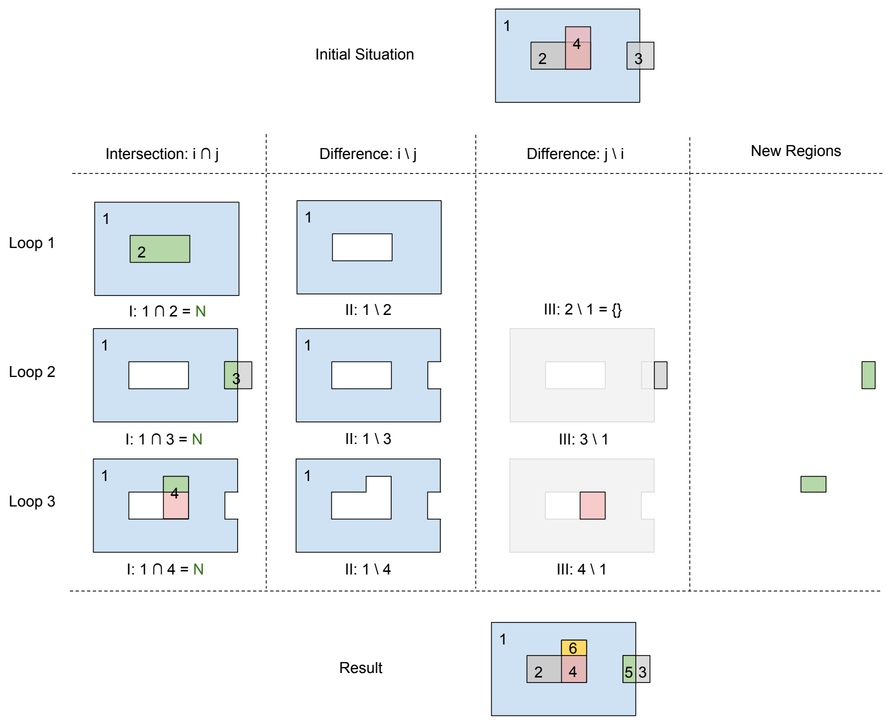


Figure 5.6: Illustration of the Advanced Region Determination Algorithm. This figure shows the sequence of intersection and difference operations to obtain all subregions generated through overlaps. The first rectangle denotes the initial situation. Loop 1-3 shows the effect of the iterations in the inner loop of the algorithm. The first column (I) represents the intersection of the regions. (II) denotes the difference of two regions, while (III) denotes the difference of the exchanged regions.

In order to execute our algorithms on this mesh a powerful data structure is necessary. The half-edge data structure (see Section 4.2) allows to query fast and efficient boundary and neighborhood relations. With this data structure the operations necessary to build the road map graph can be easily implemented. For the subsequent steps especially the possibility to iterate over edges belonging to a triangle face, attaching properties to faces and determining the center of particular edges has been exploited.

5.2.2 Road Map Graph

In the following the free space and region determination as well as the triangulation are combined to form an expressive road map. Algorithm 2 describes the basic procedure of generating this representation.

Algorithm 2: Road Map Graph Generation

```
   Data: $map\_regions$ ... list of all map regions
   Data: $obstacles$ ... list of obstacle polygons
   Data: $map$ ... polygon representing the map size
   Result: $road\_map$ ... graph representing the environment
1  begin
2      $valid\_regions \longleftarrow map\_regions \setminus obstacles$
3      $free\_space \longleftarrow$
       $Polygon(map.height, map.width) \setminus (map\_regions \cup obstacles)$
4      $triangles \longleftarrow freeSpaceTriangulation(free\_space)$
5      $polygon\_mesh.add(triangles)$
6
7      $planner\_regions \longleftarrow$
       $regionDetermination(free\_space, valid\_regions, obstacles)$
8      $polygon\_mesh.add(planner\_regions)$
9
10     $subdivide(polygon\_mesh)$
11     $road\_map \longleftarrow generateRegionGraph(polygon\_mesh)$
12 end
```

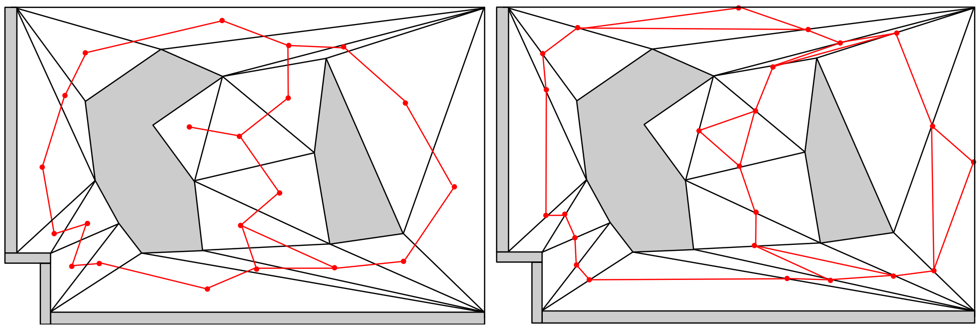
The completely triangulated environment is the basis for our road map graph. Since triangulation is one of the basic cell decomposition

methods, building road maps from this representation is quite common. A typical way of using triangulations to fabricate road maps is connecting the center of gravity (COG) of every neighboring triangle (Figure 5.7 (a)) [22, chap. 6.3.2]. A benefit of road maps generated out of triangulations is the low degree of a node. In this case the maximal number of three edges per node is limited by the geometric shape.
In our approach we want to conserve the information provided by the triangle regions. This includes the constraints of map regions and the spatial relations. Thus, we are taking the center of a triangle edge (COE) and connect it with every other COE of the triangle (Figure 5.7 (b)). Consequently we receive a connectivity graph $G = (V, E)$. G is organized as directed weighted graph. A vertex V is denoted, as already mentioned, by the center of a triangle edge and represents an entrance to a triangle, regardless if the polygon is part of a traffic region or free space. An edge E in the graph depicts the distance between a triangle entry and exit. An edge inherits the constraints attached to the corresponding traffic regions. This method leads to a degree of four per node. The degree of a node plays an important role because this has a crucial effect on the performance of heuristic search methods.
This structure of the road map provides an easy method to compute traversal costs through arbitrary regions and shortest paths in the graph. In Figure 5.8 the generated connectivity graph belonging to the running example is presented.

Region Integration

The above described road map structure allows to deal with different configurations of different traffic region types. In the following we present configurations and modifications of the connectivity graph. We show a method to integrate the constraints of various traffic region types already in the road map generation step.

Simple Traffic Regions This traffic region type is intended to provide additional properties to the corresponding region. For example to mark an area in the environment where by trend more people are moving



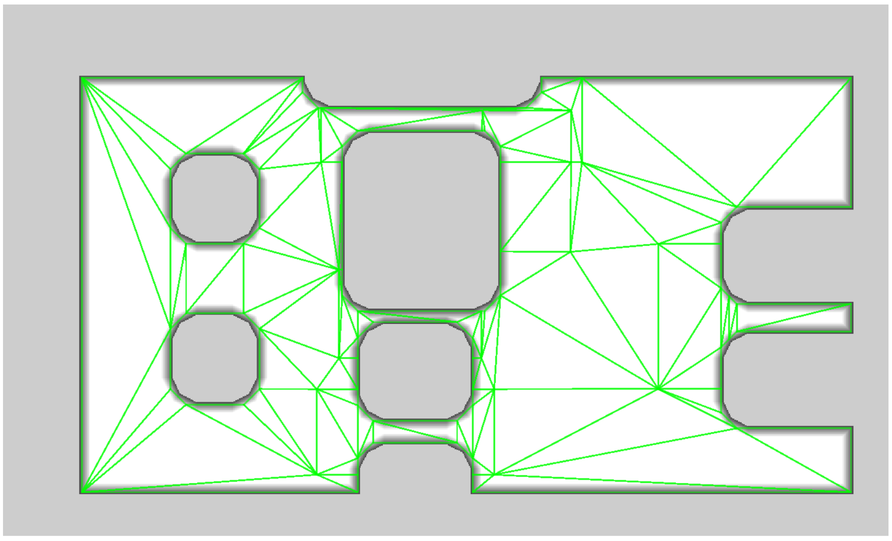
(a) Road map generated from triangle COGs

(b) Road map generated from triangle COEs

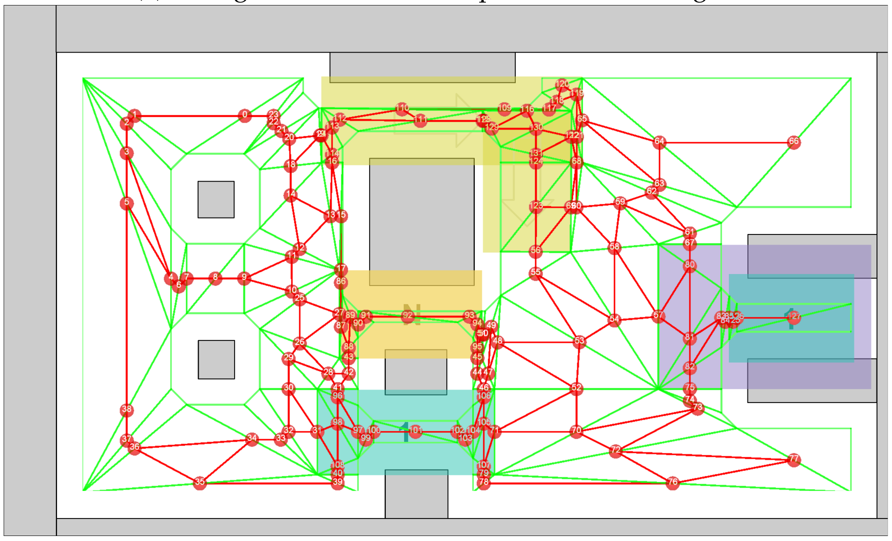
Figure 5.7: Two variants of road map generation with a triangulated polygon. (a) road map generated via connecting neighboring COGs; (b) road map generated via connecting neighboring COEs. Adapted from [22, chap. 6.3.2]

or dynamic obstacles like pallets are preserved. Therefore a caution and maximal allowed velocity value can be set.
These properties can be directly integrated into edge costs in the road map graph. Hence every edge corresponding to the region is weighted according to the properties.

Oneways A benefit of the directed graph is visible if one considers oneway (OW) traffic regions. This traffic region type constrains the traversable direction through this region. With the directed graph approach it is easy to dismiss every edge in an oneway region that points in the wrong direction. The traversal direction is defined through the angle φ between the orientation of the region $\mathbf{r}$ and the corresponding graph edge $\mathbf{e}$. $\mathbf{n}_e$ and $\mathbf{n}_r$ denote the normalized vectors of $\mathbf{e}$ and $\mathbf{r}$ (see



(a) Triangulation of the free space and traffic regions



(b) Connectivity graph overlaid with the map

Figure 5.8: Illustration of the generated connectivity graph out of the triangulation within our running example. (a) Triangulation, shown in green, of the free space and traffic regions considering the inflated obstacles (obstacles visualized with "rounded" inflation, triangulation executed on "squared" inflation); (b) Connectivity graph (red) representing the road map. Note that this illustration depicts only the connections between the nodes of the road map not the underlying directed edges.

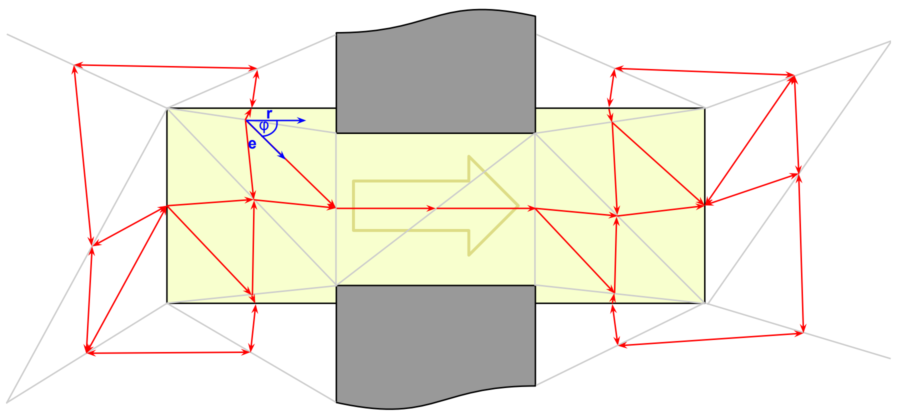


Figure 5.9: Illustration of a resulting graph for an oneway traffic region. The figure denotes the directed edges (red), represents the OW (yellow), indicates a triangulation of the regions (grey) and visualizes the angle test for one specific edge (blue)

Equations 5.1).

$$\varphi = cos^{-1}\left(\frac{\mathbf{e}\cdot\mathbf{r}}{\mathbf{n}_e\cdot\mathbf{n}_r}\right)$$
$$\mathbf{n}_e = \frac{\mathbf{e}}{\|\mathbf{e}\|}\,,\ \mathbf{n}_r = \frac{\mathbf{r}}{\|\mathbf{r}\|} \tag{5.1}$$

$$valid(\mathbf{e}) = \begin{cases} true & : 0 \leq \varphi < \pi \\ false & : \pi \leq \varphi \leq 2\pi \end{cases}$$

Right-Hand Traffic Regions In Right-Hand Traffic (RHT) regions robots should rather drive on the right side (based on their driving direction) than on the left side of a region. Thus, this type is supposed to be an extension of two opposite directed OWs. These regions are desired to bring up a better flow in highly crowded areas. Nevertheless this region allows behaviors like overtaking another robot and avoiding obstacles.
The RHT traffic type is realizable similar to the OW type. First the

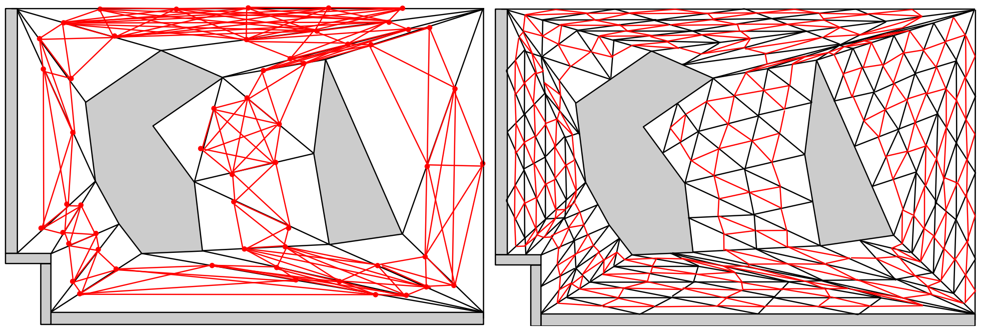
RHT is split into two regions. The region edges are treated equally to OW.

Capacity Regions and Dynamic Oneways Capacity regions, namely N-Robot and Single-Robot areas, limit the number of robots allowed to traverse a region simultaneously. Dynamic Oneways (DOW) are intended to negotiate narrow corridors or similar space configurations with multiple robots. The first entering robot determines the allowed traversal direction. If robots queue up on the other side, they have to wait until the region is completely free or an specified time interval of using the traffic region in one direction is over.
These regions do not need a manipulation of the graph structure. However they are integrated into the planning problem by considering an estimated traversal time through a region. With the estimated traversal time a time interval can be built which represents a reservation on the central timetable. How these intervals effect planning is described in detail in Section 5.3.2.

Crossings The crossing concept is also supposed to provide a better traffic flow with multiple robots. Hence, common crossing points could be traversed more fluently. A crossing works in this concept like a roundabout. In this roundabout robots up to the number of entries can be handled.
The crossing region type can be modeled in our system as multiple subregions, where every subregion handles one entrance. An individual subregion is a combination of a single-robot area and an oneway.

5.2.3 Subdivision of Triangles

Our road map generation technique creates a very sparse graph. However, this has positive and negative sides. On the one hand a sparse graph provides a small planning domain. Hence, searching on this graph is supposed to be very fast. On the other hand a road map



(a) Road map generated with longest edge subdivision

(b) Road map generated with loop subdivision

Figure 5.10: Two variants of road map generation (red) with a subdivided triangular-mesh (black triangles). (a) represents the longest edge subdivision. At is clearly visible that graph nodes of large triangles have a very high degree after this subdivision method; (b) dense road map generated via the loop subdivision procedure. This method keeps the low degree of maximal four edges connected to a node.

generated from the approach we discussed above is a coarse abstraction of the environment. Thus, paths found on this road map will not necessarily correspond to optimal paths computed by lower level planners. This is mainly a problem in the context of shortest paths, but one can argue that the high-level plan suggests only a sequence of regions and waypoints for lower level planners, which are then computing optimal paths. Nevertheless we are using topological relations to estimate arrivals and traversal times in traffic regions. If this estimations do not fit to the real path execution times the planned schedule and the corresponding reservations are inaccurate and will probably fail.

In order to counteract this problem subdivision methods are used. Subdivision is widely known in computer graphics where subdivision algorithms are used to refine and smooth polygon meshes to represent three-dimensional objects more precisely. For our need subdivision is used to generate road maps which are able to produce paths closer to optimal paths already in the high level plan. In the following we describe two methods which are generally used and applicable to our road map generation procedure.

Subdivide Longest Edge Subdividing the longest edge of a triangle is a rather simple approach. This method simply splits an edge if this is longer than a specified value. Splitting means adding an additional vertex to this edge. The left plot of Figure 5.10 shows an illustration of this method. As a result triangles obtain additional vertices. Hence these polygons are no triangles anymore. However the shape of these polygons is still equal to the previous triangles. Therefore the above mentioned advantage of visibility is still available. The resulting graph after subdividing the triangles is much denser than before. Every split edge provides an additional node in the graph. Therefore paths are smoother, less detouring and so closer to optimal paths. Nevertheless this has also a drawback: the degree of a graph node increases rapidly with the subdivision factor and the size of the triangle. As already discussed a high degree of graph nodes has disadvantages for graph search methods.

Loop Subdivision Loop subdivision was introduced by Loop [24] and is a method which splits a polygon mesh into irregular triangles introduced. In more detail this procedure replaces one element of a triangle mesh with multiple elements of the same geometric shape. Hence a single triangle is subdivided into four triangles and so forth (Figure 5.10, (b)). This method provides also the advantage of a denser graph. Moreover this approach does not change the maximal degree of a graph node.

5.3 Region Planner

Previous findings allow to produce an useful representation of the environment. The outcome is a directed weighted graph with annotated traffic region constraints. At this point we want to introduce our planning mechanisms to generate the fastest plan for a robot. Therefore the planner requests a timetable from the central server which represents reservations corresponding to the paths of other robots in the static environment. This planner is able to consider all traffic region constraints and able to respect traffic region reservations of multiple other robots. In the following a well known graph based planning algorithm, namely A* [25], has been extended in order to handle the constraints with a temporal heuristic planner. This additional planner tries to find a valid time slot in the given timetable for traversing traffic regions which does not conflict with the constraints. The last part of this section discusses the resulting plan.

5.3.1 Fastest Path Planner

The A* algorithm is a graph based planner which belongs to the group of informed search algorithms. Informed search algorithms use a heuristic function to find shortest paths between two nodes on a graph. A* is optimal and additionally complete if the heuristic function is admissible. A heuristic is admissible if the real costs of reaching a goal are never overestimated. In this case the algorithm is able to find the optimal solution, if one exists. For further reading please see [26, chap. 10.4].

As already pointed out the heuristic function plays an important role in this planning method. In the following the composition of our basic heuristic function and edge costs are described. In the road map graph presented in Section 5.2.2 nodes and edges maintain several information about their topological relations and traffic region constraints. In principle edges denote the distance between two nodes and nodes denote the location on a map. The intention of our planning approach is to find fastest paths between an initial start node and a goal node.

Hence, the edge costs have to represent the time it takes to go from a node A to a node B. In more detail if the Euclidean distance between two points Δx is known, we could estimate an approximate traversal time $\Delta t_{traversal}$, assuming a robot is driving constantly the maximum speed v_{max} allowed to drive in this region (Equation 5.2).

$$\Delta t_{traveral} = \frac{\Delta x}{v_{max}} \tag{5.2}$$

Here the traversal time is the minimal time needed to traverse the edge between two nodes. Since the road map is representing even more traffic regulating constraints than only the spatial relations, for example maximal velocities and caution values, this time has to be weighted. The constraints contribute as an increasing factor to the estimated traversal time. For further considerations the weight of an edge w_e denotes the set $w_e = \{\Delta x, t_{traversal}, t_{weighted}\}$. In this planning domain we are talking now about finding fastest rather than shortest paths in a graph.
The corresponding heuristic function to this planning problem is based on the straight line distance (SLD). In our problem we transform this function into the time dimension and call it straight line time (SLT). The $h_{SLT}(n)$ of a node n is calculated identically to the edge traversal times in the graph. The SLT is at maximum equally high than the weighted traversal time mapped to an edge. Thus, $h_{SLT}(n)$ is never overestimating the real costs and therefore an admissible heuristic. $h_{SLT}(n)$ follows also the geometrical relations of the triangle inequality (Equation 5.3), which implies that this heuristic is also consistent heuristic even in this domain. This is also illustrated in Figure 5.11.

$$\begin{aligned} h_{SLD}(n) &\leq c(n,n') + h_{SLD}(n') \\ c(n,n') &= distance(n,n') \\ h_{SLT}(n) &= \frac{h_{SLD}(n)}{v_{max}} \\ \frac{h_{SLD}(n)}{v_{max}} &\leq \frac{c(n,n')}{v_{max}} + \frac{h_{SLD}(n')}{v_{max}} \end{aligned} \tag{5.3}$$

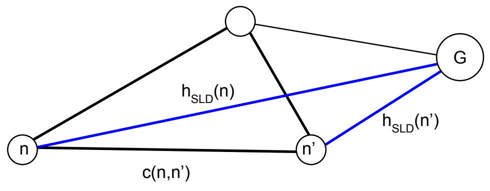


Figure 5.11: Illustration of the triangle inequality for a consistent heuristic function. The figure shows: the straight line heuristic (blue); edges in graph (black).

With this heuristic we are able to find a fastest path through the road map representing our environment. In the following an additional planning procedure is proposed. This planning method takes capacity regions into account and allows considering current region reservations from other robots.

5.3.2 Temporal Heuristic Planner

Within this section algorithms are proposed in order to include the scheduling of region reservations needed for finding a path in a set of already existing reservations already in the path finding step.

A quite simple approach to handle this problem would consider first path planning with the planning method and heuristic function explained above. The computed path has then to be analyzed whether it hits a traffic region. In case the path never goes through a traffic region the procedure has already finished. However if the path would lead a robot through a region with capacity constraints a check is necessary if the path is valid. Therefore the path costs from the start to a region entry determines the start time and the path costs between the region entry and exit the end time of a time slot the robot would need through the region. Hence it has to be checked if this time slot conflicts with the reservations on the timetable. If this is not the case the whole plan has to be rejected the corresponding edges in the graph temporary dismissed and the complete procedure inclusively path planning has to be repeated.

This approach is eventually very time consuming since every path has to be validated with the current reservation situation and every flaw requires immediately a complete new plan.

We introduce now a planning approach which could handle this problem already in the path finding step. In short our approach identifies traffic regions while planning and computes online traversal costs and waiting costs if the region is temporary not available. In addition a more accurate heuristic through that region is computed. The resulting path is used to request reservation grants for the needed traffic regions by the planning robot. Therefore the computed traversal times have to be coordinated with all robot reservations on the central timetable.

This method requires some extensions to a traditional A* planner. Algorithm 3 denotes a pseudo algorithm [27] of the widely used A* algorithm from Hart et al. [25] with the extensions. The algorithm uses two data structures: the open list is a priority queue and the closed list is a set. $openList.insert()$ inserts a node into the list, $openList.pop()$ returns the lowest element of the list, $openList.remove(n)$ removes the node from the list. The extensions to solve this problem are integrated in the algorithm and mainly located in the cost computation function. First we are looking for planning costs that represent resource allocation efforts (line 31). In Algorithm 4 these costs are modeled with a waiting time, a robot has to wait before it is allowed to enter a region. Therefore in line 2 region entrances are determined in the planning step. If a node n' is detected, which is an entrance to a region, a new temporal heuristic planner (THP) instance is invoked (line 3). The THP computes within a region the best path considering the global goal. If the THP detects a node n'' which determines a region exit a path through that region has been found. If the THP finds the goal before a region exit has been determined, the global goal lies within the region. The THP consists of two parts. The first part describes an additional A* planning instance which works in particular equally to the one described in Algorithm 3. The planner computes a complete path to either the goal, if it is inside a region, or to the exit best situated to the goal (n''). This path represents the traversal time through this region. The second part builds an interval out of the estimated arrival time at the entry node plus the traversal time and schedules an earliest

starting time (t_{EST}) considering all existing region related reservations. The waiting time $t_w = t_{EST} - g(n')$ results directly from the computed earliest starting time and $g(n')$. $g(n')$ represents the time the robot arrives at n' using the so far generated path. The method of interval scheduling is explained in more detail in Section 5.3.3.
The computed waiting time adds to the costs $g(n')$ of vertex n' and the heuristic results in $h(n') = h_{THP}(n') + h_{SLT}(n')$. The path costs through a region are represented by $h_{THP}(n')$ which is the exact heuristic. $h_{SLT}(n')$ is again the straight line time to the goal. The sum of this two parts will never overestimate the real costs. Therefore we can still guarantee admissibility and consistency of our heuristic function.
Figure 5.12 shows a scenario, which illustrates the invocation of a THP instance. In this scenario the investigation of the graph starts at the node n_{start} and chooses the node n because of the better heuristic value. At this point the algorithm identifies the next node n' to be a region entry. Thus a THP instance is started. The THP searches within the region (cyan) the best path related to the global goal and arrives eventually at node v'' which is a region exit. As a result the THP has computed the blue path between the two green spots representing the traversal time. Supposing the path states that v' will be reached in $t = 5$ and the traversal costs between n' and n'' are 4 time units, but this SR zone has been already reserved by a different robot from $t = 2$ to $t = 8$ then the plan would add to the node v' the additional waiting costs of $t_w = 3$.

5.3.3 Interval Scheduling

In this section the interval scheduling procedure is described more closely. For interval planning the relations of Allen's Interval Algebra (IA) [28, chap. 13.2] are used. IA has already been described in detail in Section 4.3. The goal of this planning component is to find the earliest possible starting time of a requested interval in a timetable.

For planning with intervals we define a query interval $Q = [q_s, q_e]$ which represents an interval lasting from a query start q_s to a query end point q_e. In the following a timetable is related to the currently

Algorithm 3: A* extensions (Adapted from [27])

```
Data: $G \ldots$ directed graph
Data: $n_{start} \in G \ldots$ start node of the graph
Data: $n_{goal} \in G \ldots$ goal node of the graph
begin
    $openList \longleftarrow \{\}$
    $closedList \longleftarrow \{\}$
    $g(n_{start}) \longleftarrow 0$
    $f(n_{start}) = g(n_{start}) + h_{SLT}(n_{start})$
    $openList.insert(n_{start}, f(n_{start})))$
    while $openList \neq \{\}$ do
        $n \longleftarrow openList.pop()$
        if $n = n_{goal}$ then
            return "found path"
        end
        $closedList \longleftarrow closedList \cup \{n\}$
        foreach $n' \in successor(n)$ do
            if $n' \notin closedList$ then
                if $n' \notin openList$ then
                    $g(n') \longleftarrow \infty$
                    $parent(n') \longleftarrow NULL$
                end
                $UpdateVertex(n, n')$
            end
        end
    end
    return "no path found"
end
Function $UpdateVertex(n, n')$
if $g(n) + w_e(n, n') < g(n')$ then
    if $n' \in openList$ then
        $openList.remove(n')$
    end
    /* calculate heuristic and waiting time                    */
    $g(n') \longleftarrow g(n) + w_e(n, n')$
    $h(n'), t_w \longleftarrow CalculateHeuristic(n')$
    $g(n') \longleftarrow g(n') + t_w$
    $f(n') = g(n') + h(n')$
    $parent(n') \longleftarrow n$
    $openList.insert(n', f(n'))$
end
```

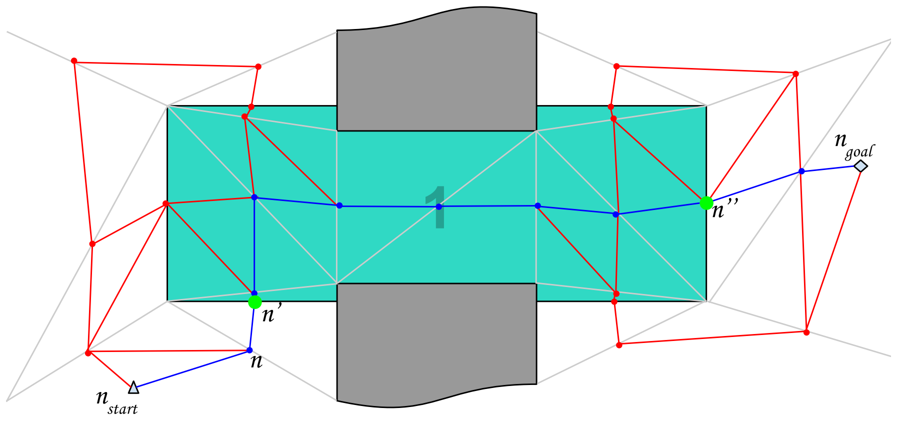


Figure 5.12: The figure presents a small graph through a single robot region. The illustration comprises: a traffic region (cyan); the connectivity graph (red); obstacles (gray); path through the SR region (blue) with entry and exit nodes (green). A* will eventually choose n for expansion, n investigates the node n'. From the entry node a path is computed through the traffic region until an exit (n'') has been determined.

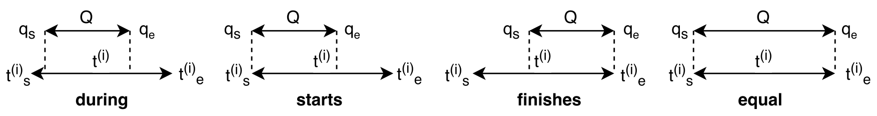
Algorithm 4: Calculate Heuristic

```
Data: $G\ldots$ directed graph
Data: $n_{goal} \in G\ldots$ start node of the graph
Input: $n' \in G\ldots$ next node
Output: $h(n')\ldots$ heuristic value of the next node
Output: $t_w\ldots$ waiting time for the next node
1 Function $CalculateHeuristic(n')$
2 if $isRegionEntry(n')$ then
3     $h_{THP}, t_w \longleftarrow TemporalHeuristicPlanner(n')$
4     $h(n') \longleftarrow h_{THP} + h_{SLT}(n'')$
5 else
6     $h(n') \longleftarrow h_{SLT}(n')$
7     $t_w \longleftarrow 0$
8 end
```

processed traffic region. A timetable is considered as set of intervals $T = \{t^{(1)}, t^{(2)}, \ldots, t^{(N)}\}$ with $t^{(i)} = [t_s^{(i)}, t_e^{(i)}]$ where N denotes the current number of reserved intervals in a timetable.

For planning we consider three queries defined with the IA relations from Table 4.1:

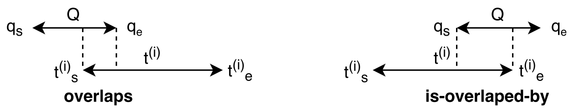
- *WITHIN*(Q, T):
  This query uses a composition of the relations $\{during,\ starts,\ finishes,\ equals\}$.

  When considering multiple time intervals with a within-query this results in a set of all intervals met by the query (equation 5.4).

  $$WITHIN(Q, T) \rightarrow \{t^{(1)}, \ldots, t^{(k)}\}, t^{(i)} \in T, 1 \le i \le k \quad (5.4)$$

- *OVERLAPS*(Q, T): This query is represented by the Q *overlap* $t^{(i)} \in T$ relation from IA.

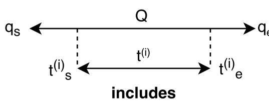
  

  When considering multiple time intervals with a overlaps-query this results again in a set of all intervals met by the query (equation 5.5).

  $$OVERLAPS(Q, T) \rightarrow \{t^{(1)}, \ldots, t^{(k)}\}, t^{(i)} \in T, 1 \le i \le k \quad (5.5)$$

- *CONTAINS*(Q, T): This query is represented by the Q *includes* $t^{(i)} \in T$ relation from IA.
  When considering multiple time intervals with a contains-query this results in a set of all intervals met by the query (equation 5.6).

  $$CONTAINS(Q, T) \rightarrow \{t^{(1)}, \ldots, t^{(k)}\}, t^{(i)} \in T, 1 \le i \le k \quad (5.6)$$



Within all queries the earliest starting time respecting a single timetable interval results in the end time of the met interval: $t^{(i)}_{EST} = t^{(i)}_e$.

Algorithm 5 presents the planning approach with the query and timetable intervals. This algorithm tries to schedule Q to get an optimal earliest start for the query interval considering all timetable intervals. In line 2 the algorithm tests if the query fulfills the constraints of the current traffic region. In case of not fulfilling the constraints the reason for this flaw is determined with one of the three explained queries. In function `computeBestInterval()` provided by Algorithm 6, which computes a new query Q' supposed to fit the constraints. In order to guarantee that there is no new flaw arising Q' must be tested again with the above procedure. If the constraint test for the query is successful a valid t_{EST} has been found. Figure 5.13 depicts a scheduling procedure for two different capacity regions.

Algorithm 6 performs an iterative search which determines the best starting time for this query considering all timetable entries and creates a new query Q' consisting of t_{EST}.

The `fulfillConstraints()` function determines to which constraint set the current region belongs. Currently the algorithm considers two different types:

- **Capacity Regions:** In order to fulfill the capacity region constraints the number of interval intersections of the query and the timetable entries must be lower than the capacity amount (Equation 5.7).
$$|Q \cap T| < C.capacity \tag{5.7}$$
- **Dynamic Oneway Region:** In this case their is no fixed capacity but a time span in which all robots have to finish traversing this region. This is realizable through attaching the traversal direction to the time interval. Hence, the region planner is aware of the valid direction and is able to compute a valid t_{EST}.

Algorithm 5: Interval Scheduling

```
Data: $Q = [q_s, q_e] \ldots$ query interval
Data: $T = \{t^{(1)}, t^{(2)}, \ldots, t^{(i)}\}, t^{(i)} : [t_s^{(i)}, t_e^{(i)}], 1 \leq i \leq N \ldots$ $N$ timetable
      intervals
Data: $C \ldots$ constraints of the traffic region
Result: $t_{EST} \ldots$ earliest starting time within timetable
begin
    while fulfillConstraints($Q, C, T$) = $false$ do
        $result \longleftarrow WITHIN(Q, T)$
        $Q' \longleftarrow$computeBestInterval($Q, result$)
        if $Q'$ ! = $Q$ then
            $Q \longleftarrow Q'$
            /* continue with next loop iteration                    */
            continue
        end
        $result \longleftarrow OVERLAPS(Q, T)$
        $Q' \longleftarrow$computeBestInterval($Q, result$)
        if $Q'$ ! = $Q$ then
            $Q \longleftarrow Q'$
            /* continue with next loop iteration                    */
            continue
        end
        $result \longleftarrow CONTAINS(Q, T)$
        $Q' \longleftarrow$computeBestInterval($Q, result$)
        if $Q'$ ! = $Q$ then
            $Q \longleftarrow Q'$
            /* continue with next loop iteration                    */
            continue
        end
    end
    $t_{EST} \longleftarrow q_s$
end
```

If the corresponding constraints fail on the query this test invokes the interval scheduling algorithm to search for another fitting time slot.

Algorithm 6: Compute Best Interval

```
1 Function computeBestInterval(Q, R)
    Input: $Q$ ... query interval
    Input: $R$ ... resulting set of $N$ elements met by the query
    Output: $Q'$ ... query with new time interval
2   $t_{EST} \longleftarrow min_{r \in R}\{t_e^{(r)}\}$
3   $Q' \longleftarrow [t_{EST}, t_{EST} + q_e - q_s]$
```

5.3.4 Formal Properties of the Region Planner

In the following we provide propositions and proof sketches regarding the termination, soundness and completeness of the introduced algorithms. For this propositions we made the following assumptions. We assume that $T^{(i)} \in \mathcal{T}$ of a timetable $\mathcal{T}$ consists of a finite set of intervals where every intervals holds a finite end time. Furthermore we assume that the algorithms are operating on a finite directed graph with N nodes and positive weights.

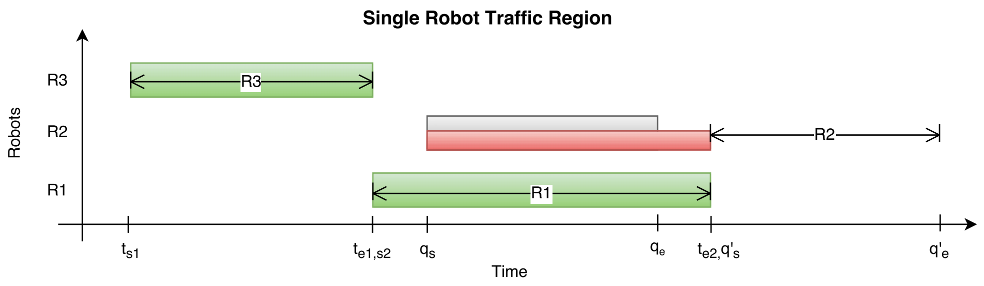
- Algorithm 6: Compute Best Interval

  Proposition 1. *Given the finite set of intervals in $T^{(i)}$ which has to be investigated for the minimization of t_{EST} the algorithm terminates with $\mathcal{O}(|T^{(i)}|)$.*

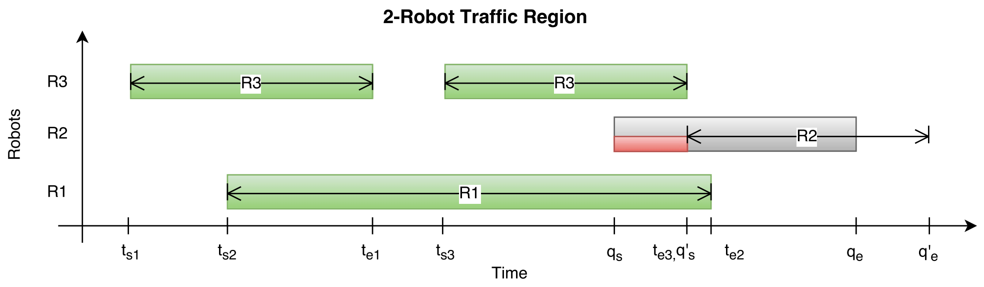
  Proof sketch. The algorithm has only one loop iterating once through the set $T^{(i)}$. □

  Given the finite end points of the intervals the result of the algorithm is the earliest possible starting time of the requested interval regarding the intervals in $T^{(i)}$.

  Proposition 2. *The Algorithm is sound and complete.*



(a) Time allocation example for a single robot traffic region



(b) Time allocation example for a two-robot traffic region

Figure 5.13: Schematic illustration of the interval scheduling procedure. (a) shows the waiting time (red) for a invalid query q (gray) and the earliest possible start time q'_s for a single robot region; (b) shows this mechanism for two-robot traffic region.

Proof sketch. This follows directly from line 2-3. □

- Algorithm 5: Interval Scheduling

  Proposition 3. *Given the finite set of intervals $T^{(i)}$ the algorithm terminates with $\mathcal{O}(|T^{(i)}|^2)$.*

  Proof sketch. Based on Proposition 2 and an absolute sortable time-sheet $T^{(i)}$ the query will be shifted by at least one interval per iteration. Therefore in every iteration the algorithm can discard one interval for further investigation. Thus the algorithm iterates at most $|T^{(i)}|$ times. Based on this bound of the iterations, Proposition 1 and the constraints are investigated at most $|T^{(i)}|$ times this proposition follows. □

  Given the finite end points of the intervals the result of the algorithm is the earliest possible starting time of the interval regarding the intervals in $T^{(i)}$.

  Proposition 4. *The Algorithm is sound and complete.*

  Proof sketch. In every iteration all intervals are computed which are intersected by the query. Based on Proposition 2 in every iteration there exists no start point, which is not in conflict, earlier than the earliest start point. It follows that the returned start point is the earliest possible starting point. □

- Temporal Heuristic Planner

  Proposition 5. *Both algorithms combined result in a run time of $\mathcal{O}(|T^{(i)}|^2 + N^2)$.*

  Proof sketch. The heuristic planner is composed of an A* algorithm and the interval scheduling algorithm. Since previously shown the interval scheduling algorithm terminates with $\mathcal{O}(|T^{(i)}|^2)$. A* is known to terminate on a finite graph with positive weighted edges in worst case with $\mathcal{O}(N^2)$ where N is the number of nodes. The THP result provides the path costs computed by A* and the waiting time computed by the interval scheduler. □

- Algorithms 3-4: Extended A*

  Proposition 6. *The Algorithm terminates with $\mathcal{O}(N^2 \cdot (|T^{(i)}|^2 + N^2))$.*

  Proof sketch. The Algorithms are using only finite functions, as previously shown, and insert and delete operations on lists which are terminating in finite time. Every iteration removes an element n from the $openList$ and adds this element to the $closedList$. n is added at most once to the $openList$, thus the list contains at most N elements. The Algorithm investigates at most all k successors of a node n. Thus the algorithm calls THP at most N^2 times. □

  Given a set of constraints, a finite timetable and a finite graph as denoted in our assumption, the algorithm finds a path satisfying all constraints if such a path exists.

  Proposition 7. *The algorithm is sound regarding the computation of paths fulfilling all given constraints.*

  Proof sketch. The algorithm generates with the help of A* a geometric path through the graph. If the path traverses a zone the THP calculates a valid interval, this follows from Proposition 4. Thus the path will never lead through a zone which is not available and therefore the algorithm is sound. □

  Proposition 8. *The algorithm is complete regarding the computation of paths fulfilling all given constraints.*

  Proof sketch. If no geometric path exists, A* terminates without finding a path, thus the proposition holds for this case. If a geometric path exists, there exists a path satisfying all constraints. This is argued with the assumption of a finite set of $T^{(i)}$ with finite interval end points. Thus one can traverse every zone after the last endpoint of all intervals. Furthermore the geometric path will be found through the completeness of the A* algorithm. □

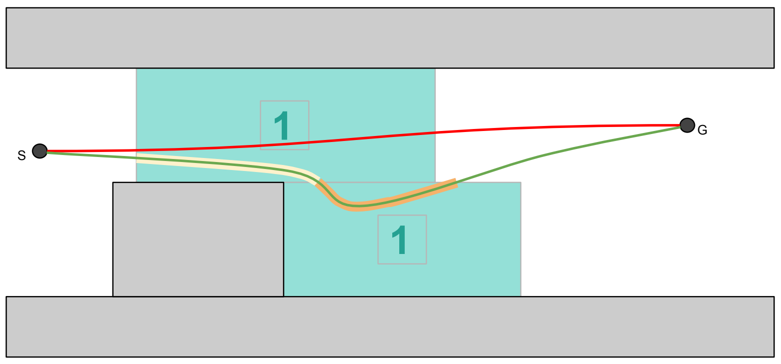


Figure 5.14: Visualization of a scenario with a non optimal plan. The scenario shows two single robot zones (blue) with forbidden areas (gray). The direct path to the goal (red) is shorter but obtains a waiting time due to reservations. The optimal path (green) respects the given reservation scenario and traverses only a part of the upper SRZ.

Concerning the optimality of the proposed algorithms the following pathologic scenario indicates that the algorithm is not optimal. This scenario consists of an environment composition similar to the illustration of Figure 5.14. Without any time constraints (reservations) our algorithms will find the optimal path, in this case the optimal path is the direct path (red) to the goal. We consider now a specific reservation configuration: there exists two consecutive reservations with a gap between them of the upper single robot zone (SRZ) and no reservation of the lower single robot zone. The proposed algorithms are still resulting in the direct path (red) to the goal, but with an additional waiting time, that allows the entrance after both reservations. In this scenario there exists a faster path. If the gap between two configurations allow traversing a part of the upper SRZ with a consecutive change from the upper SRZ to the lower SRZ and further continuing to the goal (green). This path is not computable within our algorithms since the A* in the THP finds the best region exit regarding the global goal (see Figure 5.15).

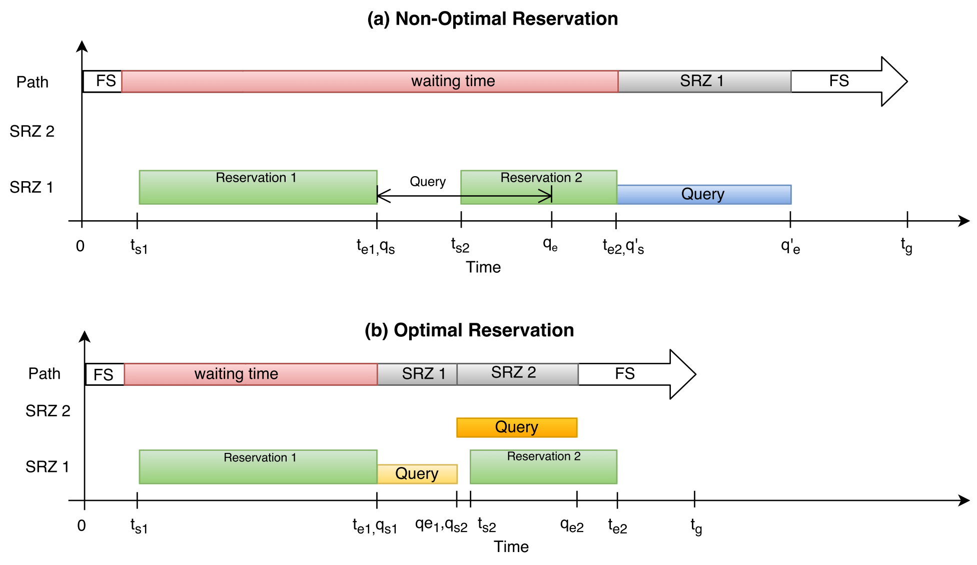


Figure 5.15: Reservation visualization of a scenario with a non optimal plan. The *Path* shows the timing of the location of the robot for the non-optimal path and the optimal path. In the first scenario (a) the robot starts in the free space (FS) and hast to wait until a query through the entire first single robot zone (SRZ 1) is valid. Then it progresses in the FS to the goal. In (b) the robot starts again in the FS and has to wait in front of the SRZ 1, but it is capable to find a faster path via SRZ 2. Finally it proceeds with the path via the SR to the goal. The second path is the optimal path, but this path cannot be generated with our algorithm.

5.3.5 Planner Result

The result of the previously described algorithms is a so called region plan. This plan consists of three parts which includes the information processed in the previous algorithms. The first part specifies waypoints for the fastest path from a start position to a goal position. These waypoints represent the entries and exits of traffic regions. The second part is a set of polygons which builds a planning window representing the sequence of regions enclosing the fastest path. The union of these polygons determines a planning window on which the grid map based planner (*Global Planner*) may plan low level paths. This represents the planning domain planning domain for the underlying robot path planning algorithm. The third part depicts a schedule for the robot. This schedule is a list of time points that suggests the arrival times of a robot at the corresponding waypoints.

In Figure 5.16 the planner result is illustrated. The figure shows a resulting plan through a free SR traffic region in our running example. Figure 5.17 shows a plan for the same start and goal combination, but this time the SR region is blocked by another robot reservation. One can see that the high-level plan respects the reservation and determines a new fastest path through the N-Robot region.

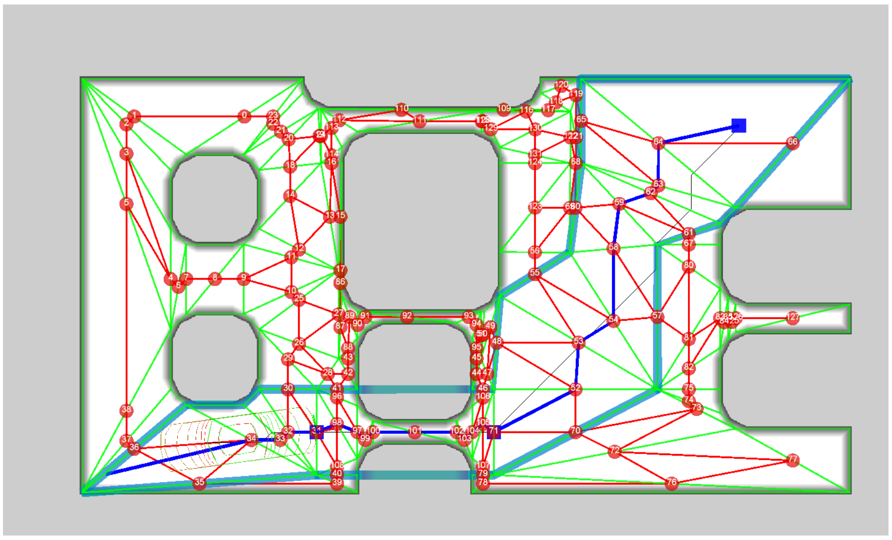


Figure 5.16: Resulting high-level plan for a start goal combination in our running example. (green) visualizes the triangulation and (red) illustrates the road map graph. The high-level plan goes through the empty single robot region: path connecting the waypoints (blue); corresponding planning window (cyan).

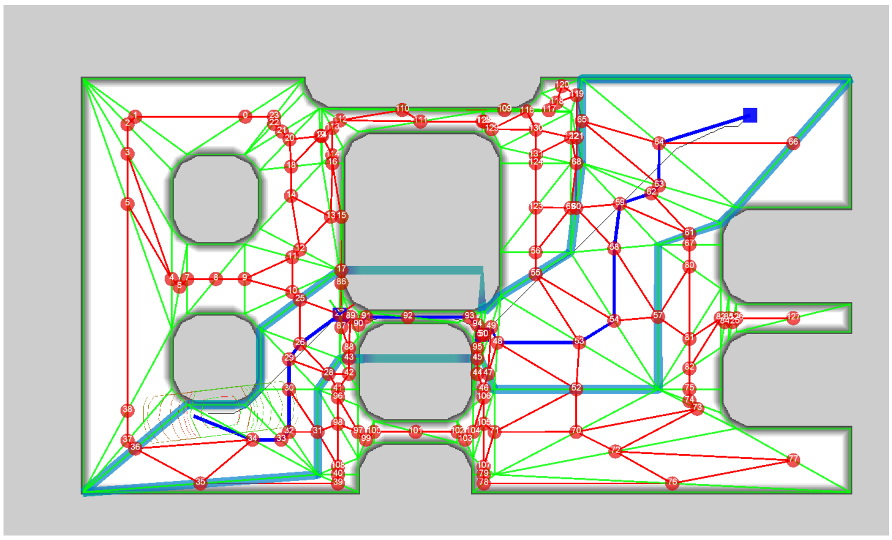


Figure 5.17: Resulting high-level plan for a start goal combination in our running example with a blocked SR region. (green) visualizes the triangulation. (red) illustrates the road map graph. The high-level plan bypasses the blocked single robot region, since waiting in front of the region would take longer than detouring the blocked region: path connecting the waypoints (blue); corresponding planning window (cyan).

5.4 Plan Integration and Execution

This part of the concept describes the high-level plan integration and execution within the hierarchical planning system. We describe first the integration and influences of the high level plan on lower level planning layers. Then the server communication modules and the timetable is discussed in more detail. Finally a validation step on the server is explained.

5.4.1 High-Level Plan Integration

Our high-level plan is directly integrated in a hierarchical navigation system based on the work of Marder-Eppstein et al. [15]. Our planning instance operates on top of a so called *Global Planner*. The *Global Planner* is in this case again an A* algorithm which plans the shortest path to a goal on a costmap. The computed high-level plan by the *Region Planner* is supposed to give a direct input to the *Global Planner*. It computes based on the waypoints and the restricted planning polygon a continuous path to the goal. The global planner path is executed from the *Local Planner*. This instance computes command velocities with a trajectory roll-out approach. These commands enables the robot to follow the path. If the robot is closely in front of a waypoint of the high-level plan, it is controlled using the schedule of the region plan. Here it is checked if the robot is allowed to enter the region, has to wait or missed a time slot. If the robot is in time everything is fine and it is allowed to enter the region. If the robot misses the timeslot suggested by the schedule it is forced to replan a new high-level plan, since we made the assumption that every unit keeps the schedule generated by the high-level plan. If the robot arrives at an intermediate waypoint earlier than expected it has to wait for the same reason. In this case we provide an opportunity to ask the *Timetable Manager* if it is allowed to enter the region earlier. In Figure 5.18 a conceptual overview of the new elements in the navigation system is provided.

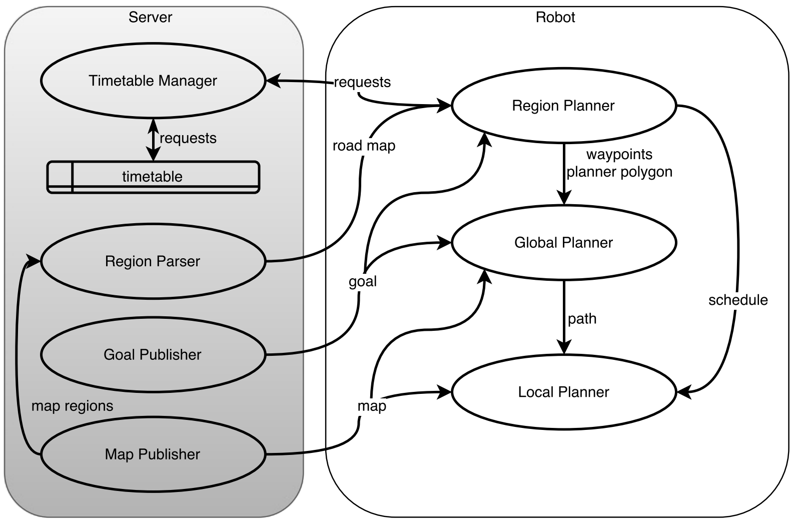


Figure 5.18: Conceptual overview of the navigation system extensions cooperating with the existing navigation stack.

5.4.2 Server Communication

The navigation system works in principle decentralized. This means that every robot is capable to navigate on his own through the environment. The central server in this system is responsible for map distribution and to distribute high-level goals, like orders or simply planning goals. With the extension of the region planner an additional shared central resource is necessary. This resource is a timetable. In detail the timetable includes for every region a separate timesheet with intervals representing the reservations of robots (Figure 5.19). The communication with the server can be divided into four requests.

Get Timetable and Reserve Time Slot Request

In the following the procedure for getting and reserving a robot time slot is presented. A robot pulls a snapshot of this timetable every time it starts to create a region plan. In Figure 5.20 the concept corresponding to the get timetable request is presented. This locally stored timetable

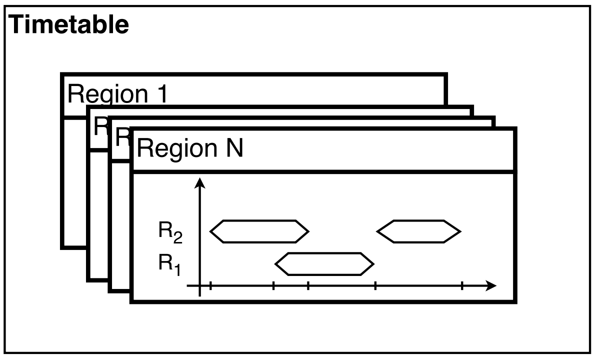


Figure 5.19: Schematic visualization of timesheets in a timetable. The timetable holds multiple region related timesheets. A timesheet keeps all robot intervals

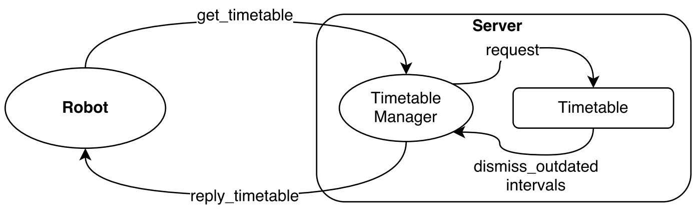


Figure 5.20: Concept of the get timetable server request.

is used to generate the plan in the way described in Section 5.3. Out of the generated schedule the robot tries to request grants for these time slots on the server. If the server validates the time slots the procedure is successful. If the server rejects the reservation the planner module has to repeat these steps. Here we have to deal with an semi-statical environment. Thus it is unlikely that two or more robots are planning simultaneously with exactly the same timetable snapshot but it is not impossible. In this case the requested time slot of the robot could lead to inconsistencies of the central timetable and has to be rejected. Figure 5.21 depicts the procedure for the reserve time slots server request.

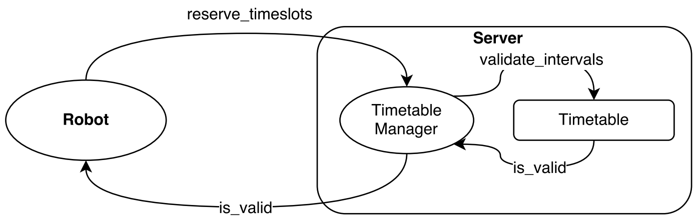


Figure 5.21: Concept of the reserve time slots and early entry server request.

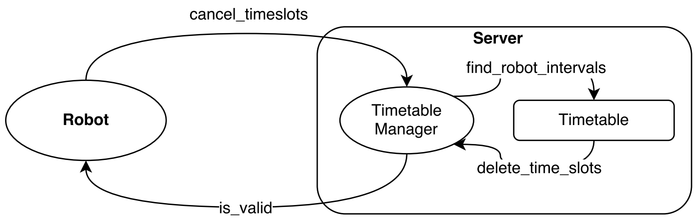


Figure 5.22: Concept of the cancel robot time slots request.

Cancel Robot Reservation Request

During the high-level plan execution there can be some reasons a robot is not able to fulfill his schedule. This could be affected by obstacles which are not represented in the environment, but also through human coworkers or even other robots. However if a robot is late and misses a reserved time slot, the complete plan has to be dismissed and the robot has to replan. Figure 5.22 shows the procedure for canceling the remaining reservations of the canceled plan of a robot on the server.

Early Region Entry Request

When executing the high-level plan produced by the *Region Planer* the system usually follows the schedule. However if the time estimation is not correct or the robot is for some circumstances at a region entrance earlier than expected a early entry request is called. This request includes the current time point and the original reserved time interval for the traffic region. The *Timetable Manager* on the server evaluates

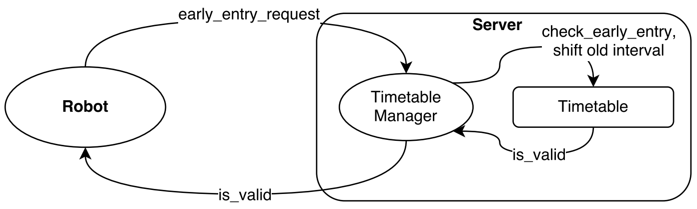


Figure 5.23: Concept of the early entry server request.

if the robot is allowed to enter the region and replies with a grant or rejection. In Figure 5.23 the concept of this mechanism is shown.

5.4.3 Central Server Validation

The central server maintains the central timetable and coordinates the requests of all robots. This module is called the *Timetable Manager*. It is also responsible to keep the integrity of the timetable after reservations have been requested and granted. The need of an additional validation step is necessary, since robots may request the timetable simultaneously. The computation of a high-level plan with a snapshot of the server timetable suggests conflicting intervals. However locking the timetable and providing access only to a single robot simultaneously is not an option because this would slow down the reservation procedure dramatically and is not scalable to large robot teams. Thus, the *Timetable Manager* checks every reservation whether it fits to the traffic region constraints of the requested traffic region. The constraints test is in principle similar to the test described in Section 5.3.3 beside there is no interval scheduling necessary in this step. Additionally to keep the timetable compact all outdated intervals on the timesheets are removed automatically.

6 Implementation Details

This chapter gives some implementation details related to this work. Our algorithms have been implemented with the ROS framework[1], described in Section 4.1. It starts with a description of the *RegionParser* module. Then details of the *RegionPlanner* are presented and finally a more detailed view is given on the integration and controlling step of the generated high-level plan in an already existing navigation module for autonomous transport robots.

6.1 Region Parsing

In the following the implementation details of the *RegionParser* node are described. This module is responsible to generate the road map graph out of a list of map regions. It is supposed to be run on the central server. Figure 6.1 shows the class diagram of the node. In this diagram the most important functions are listed. The listed functions provide basically the functionality described in Section 5.2. The main part of this module is realized by a thread, which generates a graph representing the environment. A new computation is triggered and generates a new graph on every change of either the map regions or any parameter which influences the resulting graph. Therefore the node listens on the `/central_server/map_regions` topic where every change of a map region is published. A map region consists of several information which is used to produce the road map graph. This information is listed in Listing 6.1.

[1]http://wiki.ros.org/

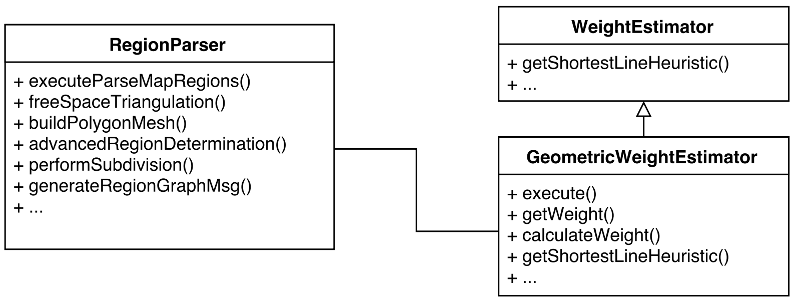

RegionParser
+ executeParseMapRegions()
+ freeSpaceTriangulation()
+ buildPolygonMesh()
+ advancedRegionDetermination()
+ performSubdivision()
+ generateRegionGraphMsg()
+ ...
WeightEstimator
+ getShortestLineHeuristic()
+ ...
GeometricWeightEstimator
+ execute()
+ getWeight()
+ calculateWeight()
+ getShortestLineHeuristic()
+ ...

Figure 6.1: Overview of the classes necessary for the RegionParser module.

In the graph building step (`generateRegionGraphMsg()`) the weights get computed by the *GeometricWeightEstimator* class. This class parses all properties from the edge related traffic regions and builds a corresponding weight out of it. The output of this function is the complete road map graph in a ROS message format (Listing 6.2). This message is published on the topic: `/central_server/region_parser/region_graph`.

Listing 6.1: MapRegion ROS message

```
int32 region_id
int32 region_type                                   # traffic region type
incubed_msgs/Footprint footprint                    # polygon points
string name
string description
float32 max_velocity                                # maximal allowed velocity
int32 allowed_robots                                # number of allowed robots
int32 caution                                       # dangerousness of a region
incubed_msgs/MapSubRegion[] sub_regions             # included subregions
  int32 sub_region_id
  incubed_msgs/Footprint footprint                  # polygon points
```

Listing 6.2: ROS message of the road map graph

```
# A message representation of a graph for the region planner module

float32 map_resolution
incubed_msgs/MapRegion[] map_regions                # List of all map regions
region_planner_msgs/Region[] regions                # List of all regions
region_planner_msgs/GraphNode[] nodes               # List of graph nodes
  uint32[] region_ids                               # adjacent traffic regions
  region_planner_msgs/Point2D position              # position of the node
  region_planner_msgs/Point2D[] line                # node related polygon edge

region_planner_msgs/GraphEdge[] edges               # List of graph edges
  uint32 region_id                                  # polygon region id
  uint32[] node_ids                                 # connecting nodes
  float64 distance                                  # eucledean distance between nodes
```

```
float64 cost                                # computed weight of the edge
float64 max_vel                             # maximal allowed velocity
```

6.2 Region Planning

The *RegionPlanner* node is the main part of this work. In this node the road map graph of the *RegionParser* is parsed into a directed graph data structure and the planning invoked by the `regionPlanThread()` in the *MoveBase* module (see Section 6.3). Figure 6.2 gives an overview of the acting classes and their most important functions.
In the first step the graph message is parsed into an extra data structure. This structure keeps adjacency lists and allows to derive immediately the successors of a node. The adjacency list represents the outgoing edges and keeps the weight information to every connected node. This is not directly possible in the ROS message format, since we have to publish a separate edge and node list because of missing object oriented features in the ROS message types. The directed graph is implemented in our *DiGraph* library.

The generation of a new *RegionPlanner* plan follows the procedure illustrated in Figure 6.3. After an invocation the first step is to catch a current timetable snapshot from the central server (see Section 5.4.2). The current timetable and the *DiGraph* data structure is transferred to the *TAstar*. The *TAstar* module provides the planning functionality described in Section 5.3.
In the planning step again the *GeometricWeightEstimator* comes into play. This class provides the functionality to compute the SLT heuristic function. Additionally in the *TAstar* module the *TemporalWeightEstimator* computes the temporal costs through a traffic region and the more accurate heuristic through that region. Therefore the *TemporalWeightEstimator* holds an instance of the *TemporalQueryProvider* which provides the methods to perform the interval scheduling algorithm. By taking a closer look at the class diagram, one can see that the *TAstar* has an aggregation to the *TemporalWeightEstimator* and vice versa. The reason for this pattern is the extra planning instance which is invoked by a

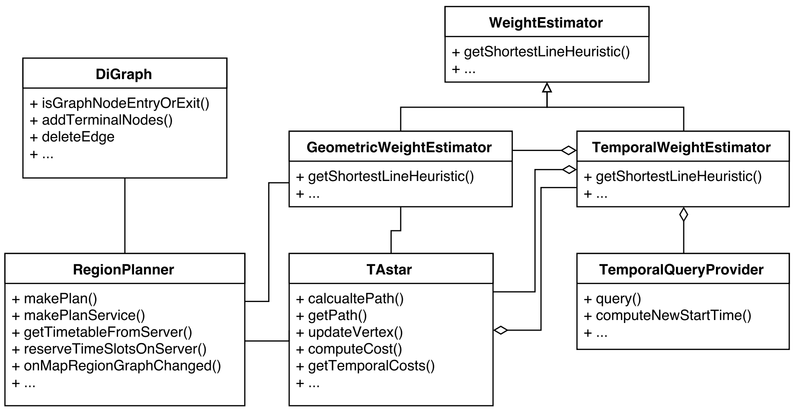


Figure 6.2: Conceptual class diagram of the region planner module. This diagram includes the most important functions of the planning module.

region entry. The *TemporalWeightEstimator* class in the primary *TAstar* instance is called if a new region is entered. Withing The *TemporalWeightEstimator* another *TAstar* instance is triggered to compute the fastest path within this region.
Finally the computed high-level plan is communicated with the central server.

6.2.1 Boost R-Tree

The underlying data structure for the interval scheduling functionality is provided by the Boost R-Tree library[2]. A R-Tree is a multidimensional spatial index data structure which can be used to represent geometric objects. R-Trees are balanced index structures and allow fast and efficient queries on relations between geometric objects. For further reading please consider [29].
The implementation in the Boost library provides queries like *overlaps, intersects, contains, within, disjoint* and some more. We use this data structure to represent our time intervals and their relations formalized by AIA (see Section 5.3.3). An interval is in this context a rectangle

[2]`http://www.boost.org/doc/libs/1_60_0/libs/geometry/doc/html/geometry/reference/spatial_indexes/boost__geometry__index__rtree.html`

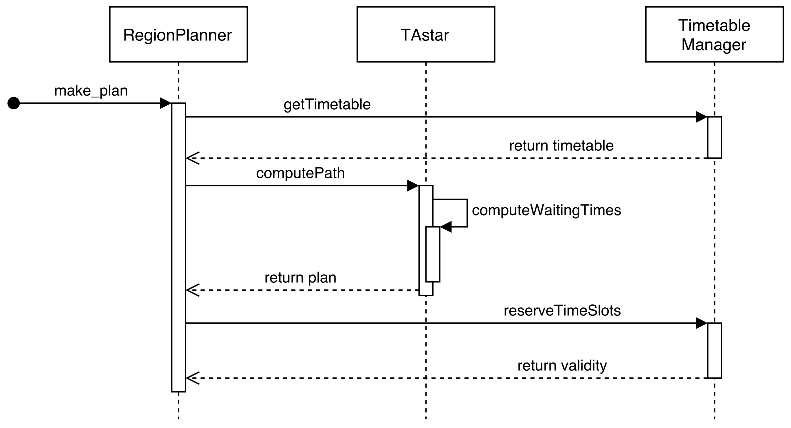


Figure 6.3: Illustration of the planning procedure in the RegionPlanner module. This diagram shows only the conceptual process of this module.

with zero width and a length representing the interval duration. The position of the lower left corner of the rectangle denotes the start point of the interval. We benefit from the index based data structure and are able to determine fast and efficient intersection points and the amount of overlaps between intervals.

6.2.2 Server Requests

In order to generate a high-level plan a timetable snapshot of the current timetable held by the server is necessary. This timetable is returned by the `/get_region_timetable` service provided by the *RegionTimetableManagerRos*. The response of this request includes all timesheets in the way presented in Listing 6.3.

Listing 6.3: ROS service message of the get timetable request

```
string robot                                              # requesting robot
---
region_planner_msgs/Timetable timetable                   # responded timetable
  std_msgs/Header header
  region_planner_msgs/RegionTimetable[] r_tables          # timesheet for a region
    uint32 map_region_id                                  # traffic region id
    region_planner_msgs/TimeInterval[] intervals          # list of intervals
bool success                                              # valid request
```

After a plan has been successfully created the time intervals have to be validated by the central server. This validation is necessary since the timetable could have been changed through requests of other robots. With the `/reserve_region_timetable_slot` service the schedule of the high-level plan is sent to the server. The request message includes the intervals for every computed region interval (Listing 6.4).

Listing 6.4: ROS service message for time slot reservations

```
string robot                                              # requesting robot
region_planner_msgs/TemporalPlanSlot[] time_slots         # list of time slots
  int32 region_id                                         # traffic region id
  int32 region_type                                       # traffic region type
  region_planner_msgs/TimeInterval[] intervals            # desired intervals
---
bool success                                              # valid request
```

Additionally the *RegionTimetableManagerRos* provides the `/cancel_region_timetable_slot` and `/request_early_entry` service. The *RegionPlanner* performs also service calls invoked by the *MoveBase* node (see Section 6.3). The cancel region timetable slots service cancels all intervals of a robot. The early entry request asks the server if an already reserved time interval is allowed to be moved to a previous time point.

6.2.3 Server Communication and Validation

The *RegionManagerRos* on the central server holds a *RegionTimeTableManagerRos* instance which manages all timetable interactions. These include following requests:

- `/get_region_timetable`
- `/reserve_region_timetable_slot`
- `/cancel_region_timetable_slot`
- `/request_early_entry`

The individual requests are handled by the *RegionTimeTableManager* class. Figure 6.4 shows the relationship of the interacting server classes as well as the most important functions.

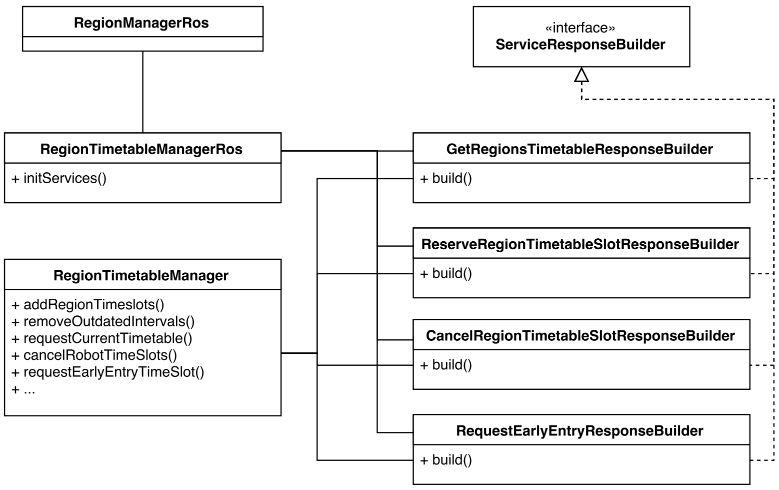

RegionManagerRos
«interface» ServiceResponseBuilder
RegionTimetableManagerRos
+ initServices()
GetRegionsTimetableResponseBuilder
+ build()
ReserveRegionTimetableSlotResponseBuilder
+ build()
RegionTimetableManager
+ addRegionTimeslots()
+ removeOutdatedIntervals()
+ requestCurrentTimetable()
+ cancelRobotTimeSlots()
+ requestEarlyEntryTimeSlot()
+ ...
CancelRegionTimetableSlotResponseBuilder
+ build()
RequestEarlyEntryResponseBuilder
+ build()

Figure 6.4: Acting classes of the timetable management. This diagram includes the most important functions of the timetable validation and management.

As already mentioned the server is responsible for the timetable integrity. This is important if two robots are planning simultaneously on a snapshot of the central timetable and their plans are using the same traffic regions. In order to synchronize the requests on the server and provide a valid timetable every request has to be locked for the duration of the request. Furthermore every reservation request is checked beforehand on a copy of the timetable. This copy serves as consistency table, if the desired intervals are valid the reservations are integrated in the real timetable. Additionally the timetable keeps itself compact by deleting outdated intervals automatically.

6.3 System Integration and Controlling

System integration is realized in the *MoveBase*[3] module which is part of the ROS navigation stack[4]. In the class diagram (Figure 6.5) and in Figure 6.6 one can see the extensions of the existing *MoveBase* module.

[3]http://wiki.ros.org/move_base
[4]http://wiki.ros.org/navigation

The *MoveBase* consists of three planner instances: *RegionPlanner*, *BaseGlobalPlanner*, *BaseLocalPlanner*. These planner instances are responsible for planning a valid path, which is coordinated with all other robots, and computing command velocities which will not drive the robot into any obstacle. A robotic system has to react on dynamic changes in the environment produced by robots or other circumstances. For this reason it is necessary to provide periodically updated navigation plans and command velocities. This requirement suggests an individual thread for every planning instance. In the available *MoveBase* this has been already realized with the `planThread()` and `executeCallback()`. The `planThread()` is intended to handle the global path planning procedure while the `executeCallback()` represents the callback of an action server. This callback is in principle a thread and is intended to trigger path planning and to execute the computation of command velocities corresponding to the computed global planner paths. In addition to this structure a `regionPlanThread()` is introduced for the *Region Planner* instance.
The thread related to the *Region Planner* produces periodically high-level plans. This way of planning creates plans fitting to the current position of a robot and current timetable reservations, even if robots have to perform any obstacle avoiding behaviors. The high-level plan provides intermediate waypoints, a planning window which is restricting the search space for path planning with the global planner and a schedule which represents the time intervals of the traffic region reservations. It is integrated and interacts in the way described in Section 5.4.1.

6.3.1 Planning Procedure

For goal execution the *MoveBase* provides an action server. The action server of a robot X is callable via the action (`/robot_X/move_base`). If this action is called a goal pose is provided. This goal triggers the planning threads to generate new plans. In the following the procedure of generating navigation plans is explained. Additionally Figure 6.7 illustrates this procedure as a sequence diagram.

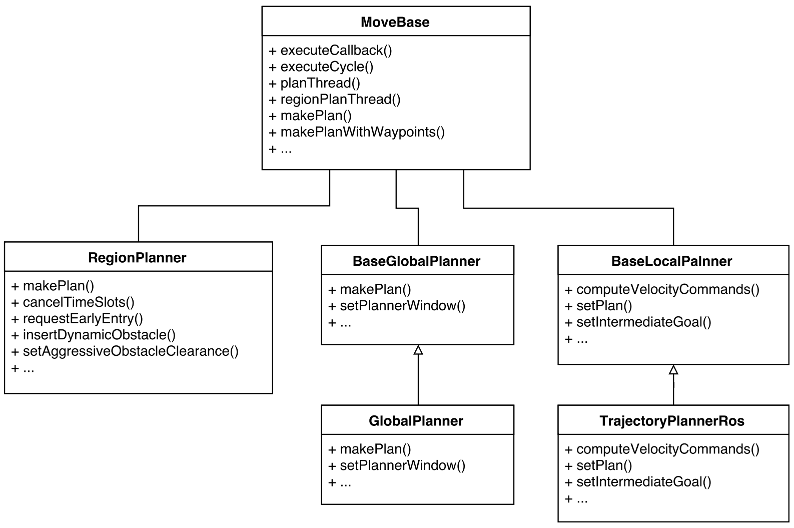


Figure 6.5: Illustration of the RegionPlanner classes interacting with the MoveBase. This diagram shows only the function necessary for integration and controlling of the high-level plan.

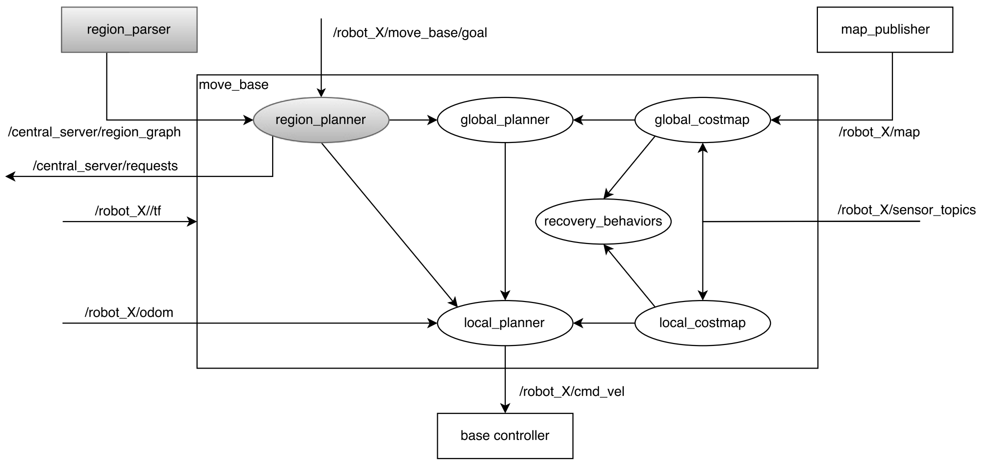


Figure 6.6: Overview of the composition of the extended MoveBase package. (gray) components are novel parts necessary for region planning. Adapted from `http://wiki.ros.org/move_base`

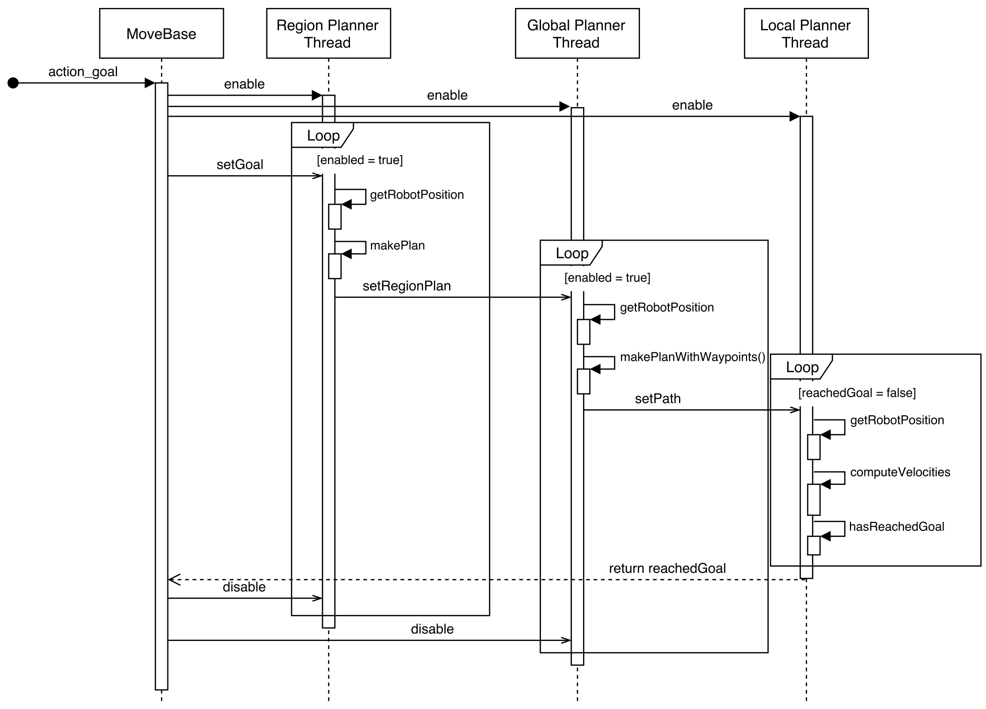


Figure 6.7: Illustration of the interaction procedure of the MoveBase integrated with the RegionPlanner. This diagram shows only the conceptual process of the module.

If a callback receives a goal first the region planner thread is notified that there is a new goal available. The `regionPlanThread()` requests the current robot position and triggers the *RegionPlanner* to produce a high-level plan out of the robot pose and the goal pose. If the region planner accomplishes generating a new plan the `planThread()` is notified with the high-level plan.

The `planThread()` is responsible for requesting paths from the *GlobalPlanner* plugin. In order to cope with the intermediate goals and the planning window of the high-level plan we had to modify the `planThread()`. The extended `planThread()` provides a possibility to trigger the generation of a path between two waypoints respecting the planning window specified by the high-level plan. This plan parts are then combined to a global continuous path. In Figure 6.8 the combined

global path respecting the region-plan is illustrated.
The produced plan to reach the next intermediate waypoint is used by the callback thread to induce the *TrajectoryPlannerRos* to compute valid command velocities. The callback thread is amongst others responsible to handle goal and more important intermediate goal completion. Intermediate goals are defined by the waypoints of the high-level plan. If an intermediate goal is reached, the callback thread reviews the schedule given by the high-level plan. If necessary actions to restore a consistent schedule are initiated. This ends in three possible situations: If the robot is in time, everything is fine and the robot is allowed to enter the region. In this situation the succeeded intermediate goal is automatically marked as reached and the *LocalPlanner* processes the next intermediate waypoint of the region plan. If a robot misses its time slot the complete region plan has to be canceled in order to keep other robots' high-level plan valid. Canceling time slots triggers the `/cancel_region_timetable_slot` service and forces the region planner to compute a new plan. If the robot arrives earlier than expected at a region entry, it is allowed to query if an earlier entrance is valid with other robot reservations (`/request_early_entry`). If this request fails the robot has to wait in front of the entrance until its schedule allows to enter the reserved traffic region.
Meanwhile the `regionPlanThread()` and `planThread()` are producing periodically new plans, fitting to the current environmental situation. In more detail due to the progress on an already computed plan, the *RegionPlanner* is able to compute a new plan corresponding to the latest position of the robot and changes on the central timetable. In the `planThread()` paths corresponding to this new high-level plan are generated again. During these two planning steps the *LocalPlanner* computes still valid command velocities for the last complete plan until the *GlobalPlanner* provides a new path.

6.3.2 Update Planning Graph

The high-level plan enforces the *GlobalPlanner* to plan only paths within the given planning window. If the low-level planners (*GlobalPlanner,*

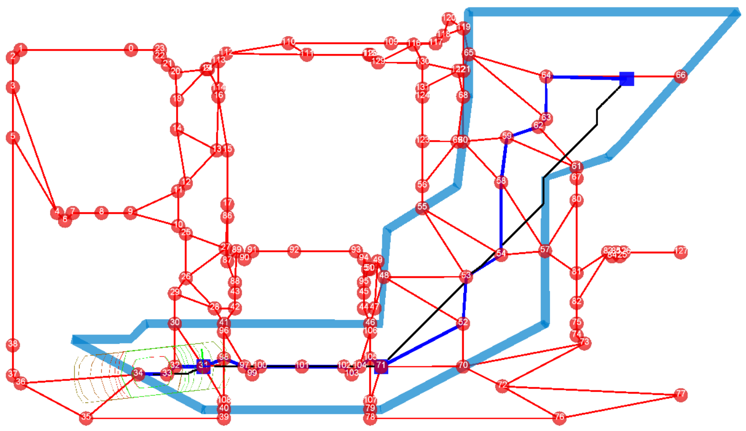


Figure 6.8: Illustration of the region-plan. The region-plan is visualized with the planning window (light-blue) and the path (blue) which connects the waypoints (blue-squares). The *GlobalPlanner* path (black) is respecting the waypoints of the region-plan.

LocalPlanner) are for some circumstances not able to compute valid paths and command velocities it is assumed that a robot is blocked by a not represented obstacle in the map, for example a temporary parked palette. In case the lower level planners report that problem in the form of an incomplete path and the location of the assumed obstacle we have to include this information into our representation. Otherwise the *RegionPlanner* would suggest equal waypoints and planning windows. Therefore we provide a functionality to avoid this issue. We introduce a possibility to determine the obstacle related polygon represented by the graph and to remove the corresponding edge which led to the inconsistency of the path for the next region planning trial. Thus the new high-level plan is aware of this obstacle in the environment. If this procedure does not help, there is the opportunity for a more aggressive behavior. If the high-level plan is not executable for a while the complete path, besides the edge related to the start and goal position, will be set invalid temporary. This behavior indicates a complete new plan avoiding the blocked path.

7 Evaluation

The following chapter presents the results of the evaluation of this work. The first part of this chapter gives an overview of the functionality of the planning concept we will use in this evaluation. The second part focuses on the used evaluation scenarios and describes properties of the used environment representation. Then the performance of the planner is investigated independently from the overall system including the multi-robot approach and the low-level navigation. This is used to evaluate the scalability of the basic planning approach. Then the performance of a multi-robot system using the introduced high-level planning procedures is evaluated and finally some use cases are demonstrated where our hierarchical planning approach yields performance gain for a multi robot system.

7.1 Overview

In this chapter we investigate the properties of the hierarchical planner extension. We showed in Chapter 5 a way of integrating different traffic region types into a hierarchical path planning system. In this chapter we show the performance of the hierarchical planning approach for a set of integrated region types. The implementation of our system processes the region types presented in Table 7.1. Region types marked with a check are fully integrated in the system. Region types with an asterisk are integrated in a conceptual way in the environment representation and the planning problem.

Region types described by this table are used to generate evaluation scenarios. This scenarios are used to evaluate the performance of the

Region Type	Abbreviation	Integration
N-Robot	NR	✓
Single Robot	SR	✓
Forbidden	F	✓
Simple	S	✓
Oneway	OW	✓
Dynamic Oneway	DOW	$*$
Right Hand Traffic	RHT	$*$
Crossing	C	$*$

Table 7.1: Integration status of the traffic region types.

new planner and the overall system. Our evaluation methods focus on two different aspects. First we investigate the performance of the new planning instance by executing the planner on environments with increasing complexity. The second aspect focuses on the execution time of navigation tasks in a multi-robot system.

7.2 Evaluation Scenario

As already described in Section 5.2 an environment may consist of multiple traffic regions with different types. In order to generate evaluation scenarios, we took the traffic region types that are fully integrated into our system, and built environments of increasingly complexity. Our assumption in this evaluation is that environments with increasing number of traffic regions provide increasing complexity. This assumption is based on the resulting graph generated from the environment representation. The higher the degree of fragmentation of the environment the more nodes are in the road map graph. As a consequence the planning domain grows. Additionally the traffic region type is supposed to have an influence of the planning problem. Since capacity regions (NR, SR) triggers the temporal heuristic planner (THP), we assume that capacity regions would also increase complexity. In Figure 7.1 we introduce three evaluation scenarios of the same map size

	# Traffic Regions	# Graph Nodes
Level 1	10	65
Level 2	20	148
Level 3	32	254

Table 7.2: Tabular overview of the created evaluation scenarios and their measurable quantities

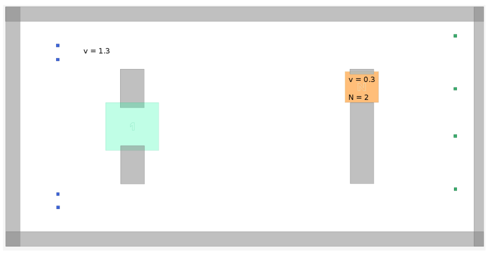
of $50m \times 100m$ but following the previous assumptions.

If one takes a closer look at the different scenarios one can see that the evaluation scenario (b) is a reduction of the most complex scenario (c) and (a) a reduction of (b). This composition is important and allows to generate the same tasks, that start and goal positions, for all three scenarios. The environments of the scenarios are built out of a frame of forbidden areas. Additional forbidden areas have been used to fragment the environment and to build narrow corridors with space for only a few robots. Corridors are filled with capacity regions to ensure that robots are not deadlocking in those corridors. Additionally in scenario (b) and (c) one way regions and simple regions are introduced. In Table 7.2 the characteristics of the evaluation scenarios are demonstrated. In this table one can clearly see that the number of nodes in the resulting graph grows with the number of regions and therefore the defined complexity.

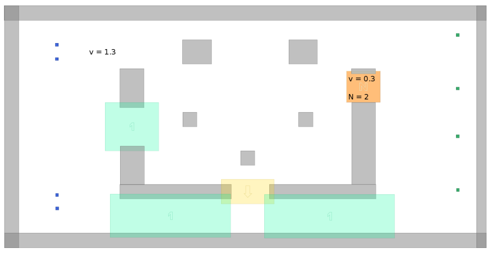
7.3 Planner Evaluation

This section aims at a performance evaluation of the *RegionPlanner* instance. The performance criteria in this evaluation is defined by the computation time needed for plan generation. With this evaluation we intend to show that the introduced heuristic in Section 5.3 is suitable for the graph search procedure with our algorithm.

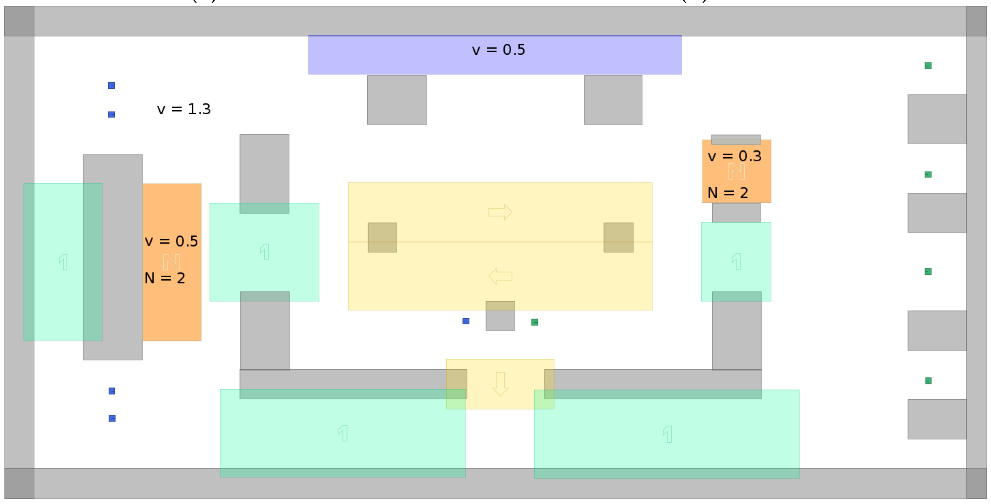
The evaluation setup consists of 1000 randomly generated pairs of start and goal positions. We sample the positions in the most complex scenario (*Level 3*). Based on the previous definition of the environment



(a) Level 1



(b) Level 2



(c) Level 3

Figure 7.1: Evaluation scenarios with increasing complexity. The variable v describes the maximal allowed velocity in the corresponding region. The blue and green spots in the free space denotes goal stations. The maps comprises: one ways (yellow); n-robot zone where maximal two robots are allowed (orange); single Robot zone (green); simple traffic areas (blue); forbidden areas (gray)

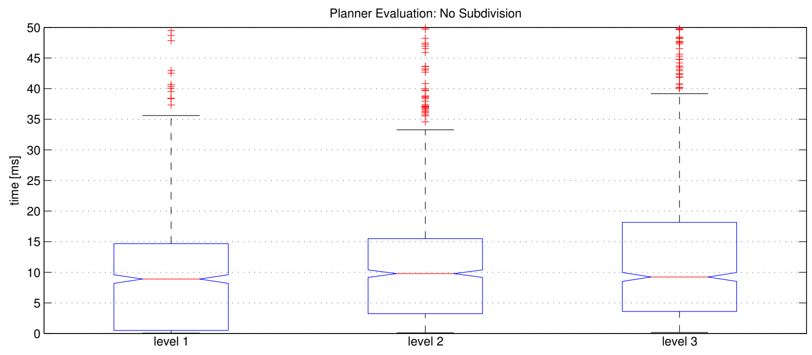


Figure 7.2: Planner evaluation on three different evaluation scenarios.

we can guarantee to sample valid start and goal positions in every scenario. Furthermore we provide equal problems for every evaluation scenario which allows for a fair comparison. In order to dismiss not significant tasks the position pairs have to have a minimal Euclidean distance of $d = 10m$.

In Figure 7.2 the evaluation results of the previous evaluation setup are illustrated. This figure shows the distribution of the computation time per evaluation scenario with a box plot. By examining these results one can see that the *RegionPlanner* is able to find plans in reasonable time. Another observation provided by this figure is that the assumed increasing complexity of the evaluation scenarios does not affect the performance dramatically. The medians of the individual evaluations are approximately equal. Even the whole data distribution shows many similarities. Since good average performance on growing graphs is the nature of good heuristics in heuristic search methods (the heuristic function ideally avoids expanding unnecessary graph nodes), the observation is not surprising. As a result on can say the chosen heuristic performs well.

7.3.1 Worst Case Analysis

The observations of the first evaluation suggests that the used heuristic is performing well on problems of this kind. However these observations raises the need of harder problems in order to quantify the worst case computation of the planner. Considering harder problems difficulties for the planning instance have to be identified. We assume that paths through reserved regions will challenge the *RegionPlanner* module. Realizing harder problems requires on the one hand plans traversing most allocatable regions and on the other hand a simulation of other robot plans. The former can be realized with a larger Euclidean distance between a start and end goal (at least the map width). Due to the scenario topology this would necessarily result in more paths going through an allocatable region. The latter is realized with randomly sampled intervals requested on the server before a plan is generated. The interval is created with a randomly generated duration and start time.
We define two reservation models: (1) fluctuating reservations, (2) long reservations. The first model creates reservations with durations sampled in the range of $[10, 200]$ seconds and start times between $[0, 10]$ seconds. Therefore intervals are generated that do not reserve regions continuously. Hence, the planner has to find a path and schedule corresponding the interval set to find a valid plan. This includes the computation of waiting times. (2) generates reservations with durations in the interval of $[100, 200]$ and start times in the range of $[0, 1]$ seconds. Thus, reservations tend to start immediately and last "long" which likely prohibits a robot to find a valid time slot for the region quickly.

In Figure 7.3 the distribution of four experiments in the scenario of *Level 3* are shown. The first box labeled with "First Eval." illustrates the previous evaluation which serves as a base line. The "No Reservation" box is already part of the worst case analysis, but this experiment is without reservations. Box "Fluctuating Res." and "Long Res." correspond to the new problem definition with interval models of (1) and (2) respectively. As one can see the performance of the planner applied on the new problems is actually worse than for the primarily defined

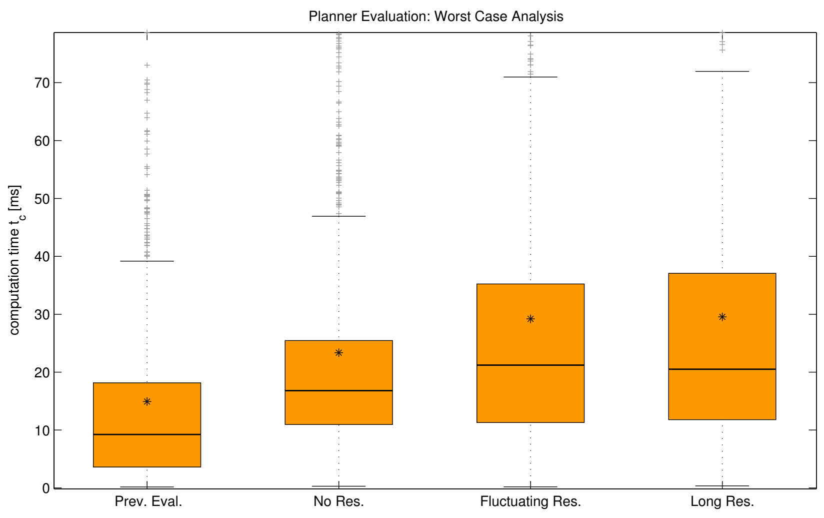


Figure 7.3: Evaluation of a worst case scenario for the RegionPlanner. The asterisk denotes the average value (computation time per goal).

problem. Although already the enlarged Euclidean distance increases the computation time. The evaluation trials considering reservations between start and goal are even worse. Furthermore it seems to be not relevant for the worst case computation whether the planner has to deal with (1) or (2). These observations indicate that we can generate the worst case for our planning algorithm by forcing the planner to investigate any region reservations and to detour allocatable traffic regions.

7.3.2 Subdivision Evaluation

A further evaluation considers the computation time of the *Region-Planner* when using the proposed subdivision methods (described in Section 5.2.3) in the road map generation phase. Unfortunately we have no direct possibility to compare the path quality of the *Region-Planner* with the plan generated by the low-level navigation module. However we assume that smoother paths, which are fitting better to

	Subdivision Degree	**# Traffic Regions**	**# Graph Nodes**
Level 1	No	10	65
	Normal		328
	Fine		888
Level 2	No	20	148
	Normal		397
	Fine		996
Level 3	No	32	254
	Normal		552
	Fine		1308

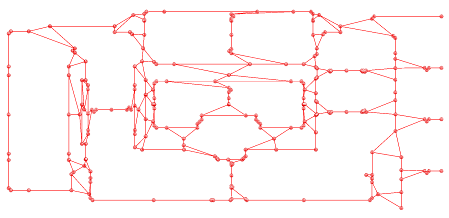
Table 7.3: Characteristics of the subdivided evaluation scenarios.

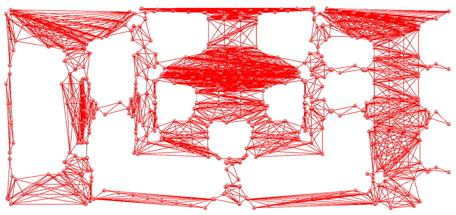
the path a robot has to follow, provide better estimations of traversal times through regions. As a consequence a more accurate resource management would be possible.

In our work we implemented the method of "Longest Edge Subdivision". The following evaluation investigates the subdivided evaluation scenarios introduced in Section 7.2 in comparison to the standard road map graph. We introduce three degrees of subdivision: *No, Normal, Fine*. No subdivision is self-explanatory. Normal subdivision splits every edge longer than a length of $l = 5m$. *Fine* applies this procedure for edges with a length of $l = 2m$. The three degrees produce different densities of road map graphs (illustrated in Figure 7.4). Table 7.3 shows the characteristics of the subdivided scenarios.

The evaluation setup consists again of the same 1000 randomly generated start goal pairs introduced in the first planner evaluation (minimal Euclidean distance of two positions $d = 10m$). The computation time t_c for a plan is measured for every goal and every degree of subdivision of the evaluation scenarios. The resulting distributions are visualized in Figure 7.5 via box plots. The individual plots (a-c) show the computation times for every evaluation scenario with increasing subdivision degree.

The data in Figure 7.5 shows clearly an increasing trend of the compu-


(a) Road map graph with *No* subdivision


(b) Road map graph with *Normal* subdivision

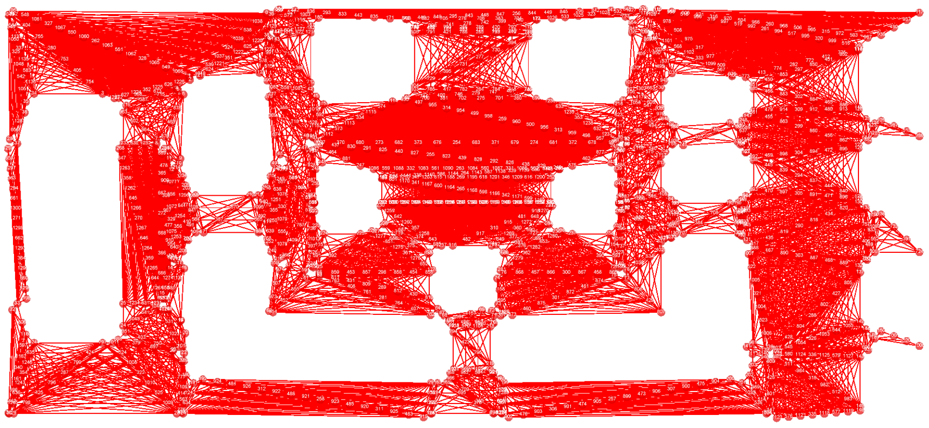

(c) Road map graph with *Fine* subdivision

Figure 7.4: Application of the longest edge subdivision method with subdivision degree *No, Normal, Fine*.

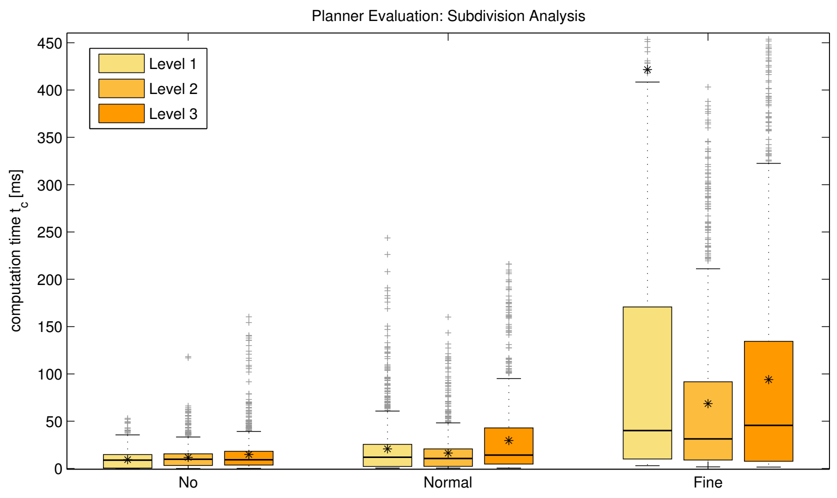


Figure 7.5: Resulting computation time distributions with different subdivision degree applied on all evaluation scenarios.

tation time for every applied subdivision degree. Especially the *Fine* subdivision creates a significant increase of the computation time for plans in all levels. In comparison to the approximate median of no subdivision levels $t_{c,median}^{(No)} \approx 10ms$, the median of *Fine* subdivision computation times is about $t_{c,median}^{(Fine)} \approx 39ms$.
Basically this result relates to assumptions made already beforehand. The longest edge subdivision method produces many additional vertices. The added vertices do not change the shape of the triangle but our road map graph generation approach takes every center of edge (COE) of a polygon and connects it with every other COE. Thus, the degree of a graph node is likely to grow fast which negatively affects the heuristic search methods. Furthermore we observed an interesting behavior by comparing the complexity levels. Interestingly a least complex scenario tend to produce the worst computation times. A closer investigation of this observation shows that the topology of the environment plays an important role for the subdivision algorithm. *Level 1* provides big areas of free space. These areas can be represented with a small number of big triangles. Since the longest edge subdivi-

sion algorithm has to split long edges more often than short edges, big triangles inherit an intensively interconnected mesh in the road map graph. Hence, environments which generates big triangles tend to have worse computation times.
Summarizing this evaluation shows that the subdivision algorithm we have implemented is working but not performing well. The second subdivision algorithm "Loop Subdivision" is supposed to be more promising for this problem. The loop subdivision method would keep the degree of nodes small but also provides a better path quality. Nevertheless there has to be made a trade-off between the computation time and a desired path quality.

7.4 System Evaluation

The second part addresses an evaluation of the overall system performance. As described in Section 6.3 we integrated the *RegionPlanner* in an existing multi-robot system which operates in an industrial logistics scenario. We evaluated this system with a simulation of the navigation system provided by the company (Figure 7.6). The system evaluation investigates the performance gain of the execution time t_e of an individual robot executing a plan and the overall completion time t_A of a fleet of robots representing a criteria for the throughput. Therefore we compare the existing "Original System" currently running on the robots with the *RegionPlanner* extensions, further referred as "New System". In this evaluation not all traffic types are considered. Here we focus on the region types also working in the *Original System*.

The evaluation is set up with 50 evaluation trials, where every trial comprises a navigation task per robot, consisting of a start and goal position randomly sampled in the *Level 3* scenario. The number of trials is in this evaluation lower, since the robots have to execute the navigations tasks in real time, thus the system evaluation takes much more time. in have to execute In this evaluation we consider fleets of $R = \{2, 4, 10\}$ robots performing individual tasks. We perform evaluations for all scenarios and the three different sizes of robot fleets

Figure 7.6: Multiple robots of the *incubedIT* system are performing tasks in a logistics scenario. ©2016 INCUBED IT - AUSTRIA

with the *Original System* and the *New System*. For the system evaluation the *Normal* subdivision degree has been chosen. As already mentioned in Section 6.3 we trigger a new plan periodically. In this setup the planner generates every $10s$ a new plan this is necessary to be able to react to changes in the environment on the real system.

Figure 7.7 shows the results of the evaluations for the different scenarios. The plots compare the execution times of the navigation tasks executed by the *Original System* and the *New System*. The left bars in the scenario groups correspond to the *New System*, right bars to the *Original system*.
The plots in the figure generally state a continuous increase of the execution times with increasing complexity of the scenarios. Furthermore the execution times rise with the number of robots in a fleet. The former observation can be motivated by the scenario topology. The more obstacles a scenario consists of the more likely longer paths which have to detour obstacles are. The latter is basically due to the fact that in multi-robot scenario the probability of colliding paths increase with the number of robots in a fleet. However the amount of time loss due

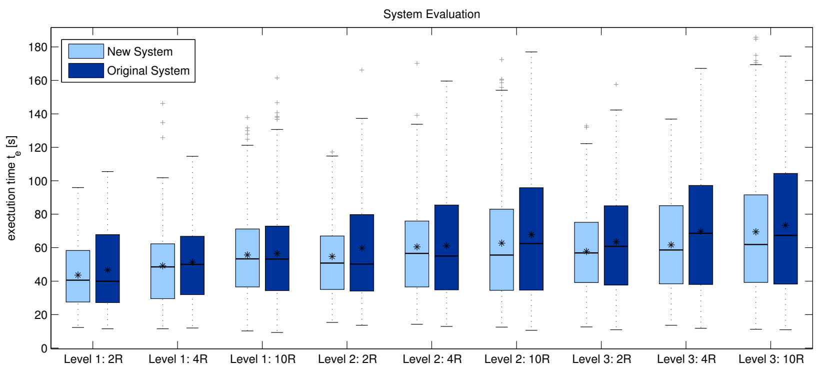


Figure 7.7: Visualization of the system evaluation with different number of robots. Comparison of the execution time (t_e) distributions of the New System (light-blue) and the Original System (dark-blue). The execution time denotes the time an individual robot need for executing the task. The asterisk (*) denotes the average value (execution time per goal).

to the path collision is obviously depending on the evaluation setup. The data of the *Original System* show in general higher execution times than the *New System*. The average value denoted as execution time per goal shows also the same characteristics. Hence this figures state that the *New System* is actually increasing the performance, but the data shows that the gain of performance is small.

For this reason we tried to generate a setup which favors the region planner and is moreover closer to a realistic scenario. Therefore we define a list of fixed goal stations in the map. These stations are denoted as blue and green spots in the evaluation scenarios (Figure 7.1). The setup consists of 50 random goals drawn from this list. We perform this evaluation only on scenario *Level 3* but again with $R = \{2, 4, 10\}$. The results of this evaluation are depicted in Figure 7.8.

This figure shows again the execution time per goal and the distribution of the measured execution times via a box plot. In this plot the performance gain of the *New System* is even higher. This is of course the desired effect, since this more realistic scenario provokes paths through allocatable regions. This paths generate the need of reservations and a high-level plan which considers paths of other robots too.

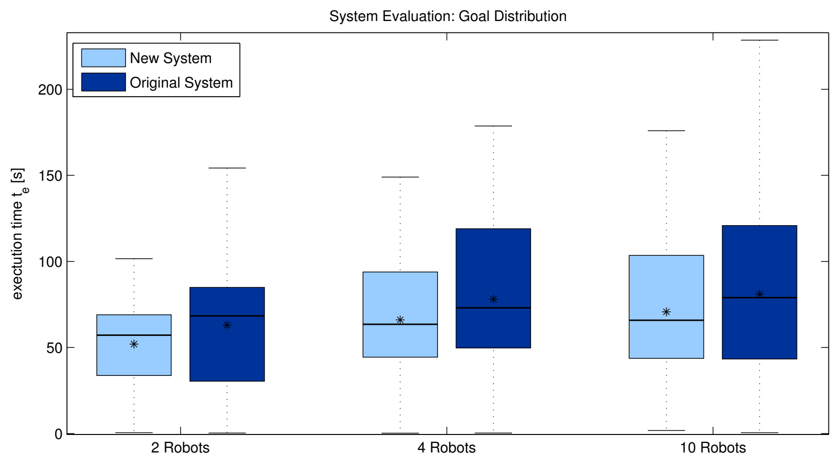


Figure 7.8: Visualization of a more realistic system evaluation with different number of robots. Comparison of the execution time (t_e) distributions of the New System (light-blue) and the Original System (dark-blue). The execution time denotes the time an individual robot need for executing the task. The asterisk (*) denotes the average value (execution time per goal).

Furthermore we investigated the throughput of this system. We define the throughput as the completion time of an evaluation trial considering the complete fleet of robots. This is actually the arrival time of the last robot at the goal of its current navigation task. Figure 7.9 shows plots considering *Level 3* with the previous more realistic evaluation setup. These plots show that even the throughput is significantly increased by the *New System*. In the evaluation runs situations appeared where a robot could not finish its navigation task. This problem is normally handled by a high-level state machine, which was not usable in the system evaluation. The problem with incomplete tasks is a disturbance of the completion time of one evaluation trial. There are probably only a few tasks leading to an incomplete goal, but the evaluation runs have been executed for 50 navigation goals per robot only. Nevertheless we had to remove the evaluation trials of an incomplete task from both the *Original System* results and the *New System* results. Otherwise a comparison of the corresponding completion times would not be fair. However this is not affecting the previous results because

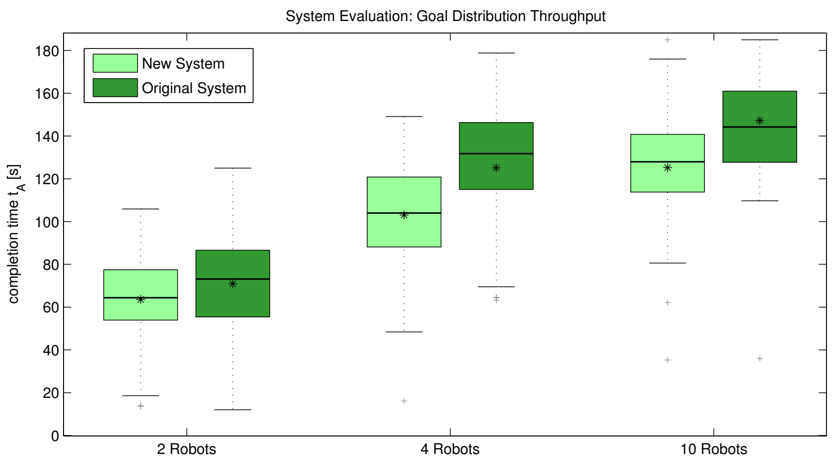


Figure 7.9: Visualization of the completion time t_A for the evaluation trials of the all robots in fleet. The completion time represents the execution time of the last finishing robot and represents the throughput.

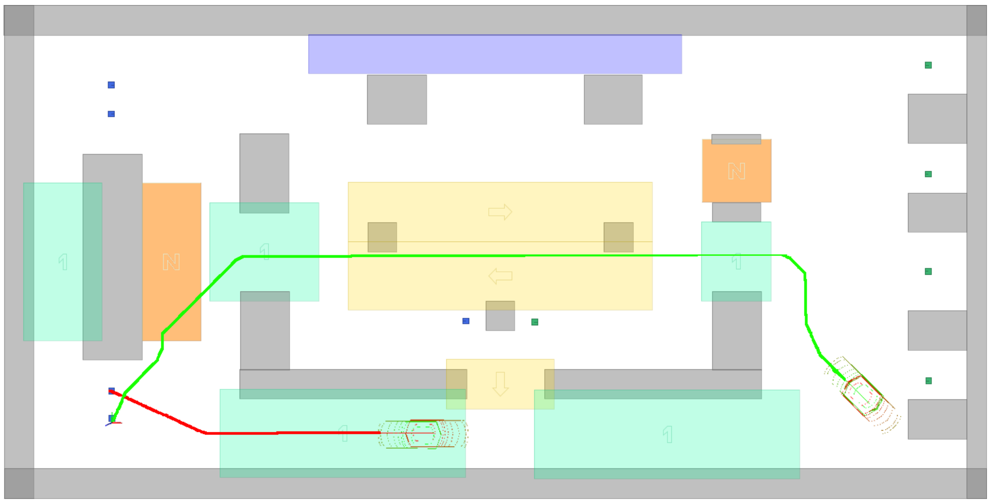
there we consider the execution time of the individual robots, which provides much more data.

7.5 Use Cases

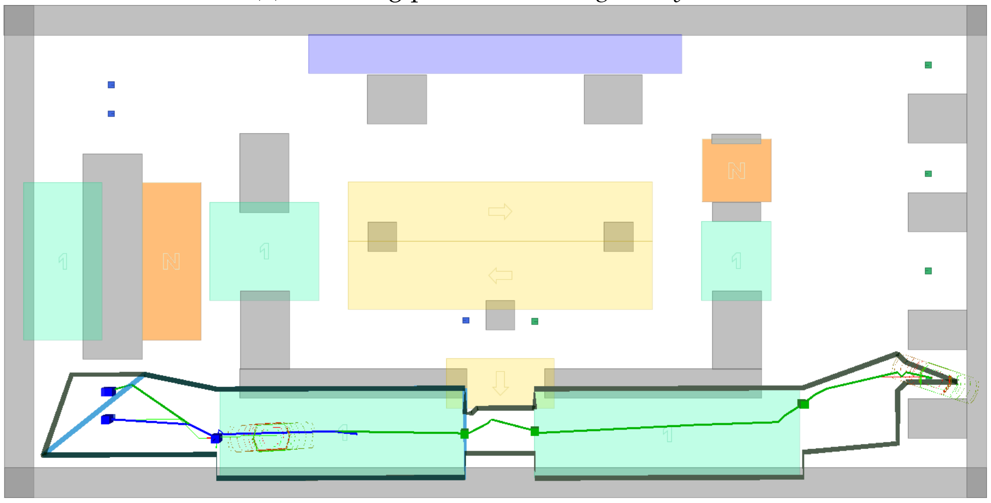
The previous evaluations suggest some use cases where the *RegionPlanner* is able to produce a performance gain. In the following we describe two exemplary scenarios presenting the effects of the high-level plans for two robots.
The use cases are demonstrated in the *Level 3* scenario. In the first use case we focus on the long corridor at the bottom of the scenario. This corridor is partitioned with two single robot areas and an additional one way entrance in the middle of the map. We sketch the following scenario: two robots are located in front of the right entrance of the corridor. The first robot R_1 receives a navigation task to a goal on the left side of the long corridor. For this robot both systems will deliver a similar path directly through the corridor since no region is reserved.

Eventually the second robot R_2 receives also a navigation task with a goal on the left side of the corridor. The first robot has already entered the second single robot zone. The path planner of the *Original System* tries to find a shortest path R_2. Due to the blocked single robot zone by R_1 this path has to detour the corridor (Figure 7.10 (a)).
The *New System* is aware of the time span the second single robot zone is reserved by R_1 and suggests R_2 to traverse this corridor too. This is reasonable since the reservations of the corresponding regions do not overlap. This plan is visualized in Figure 7.10 (b). One can see that both planning windows are overlapping. Thus the high-level plans of both robots are suggesting a traversal of the long corridor.

The second scenario demonstrates a typical goal execution similar to the more realistic evaluation shown in the previous section. This use case is sketched as follows: Two robots are located on two start stations in the upper left part of the map (see Figure 7.11). They receive simultaneously a navigation task to go to a goal in the lower right part of the map. The *Original System* plans in this case an equal path for both robots which traverses two single robot zones and additionally a one way area. However if one takes a closer look on the path visualization in Figure 7.11 (a) it is likely that both robots are arriving at approximately the same time at one of the single robot zones. Since the robots are not aware of single robot zones, they will go into the direction until one of the robots actually enters the region. Hence the single robot zone is blocked for every other robot. Thus one of them is requested to make a detour, but an earlier knowledge of the reservation would have created a better path.
The *New System* is computing a path which considers the reservations of both robots. Thus a plan is generated which can prevent this behavior beforehand. In Figure 7.11 (b) the high-level plan demonstrates how the paths of both robots are coordinated. Instead of leading both robot in front of the the single-robot zone, one robot is detoured early to guarantee still a fast path to the goal. The n-robot area is additionally avoided since this region has a velocity constraint which makes it advantageous to detour this region too.

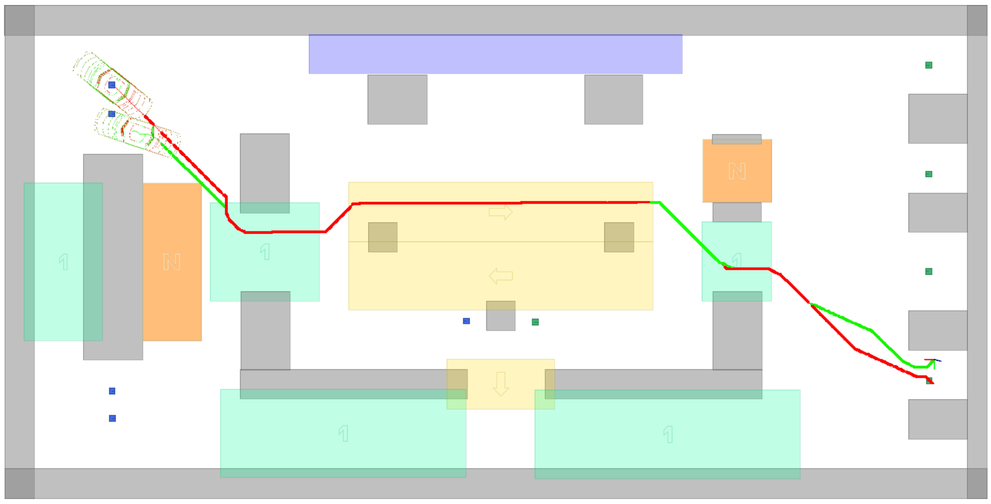


(a) Resulting paths of the *Original System*

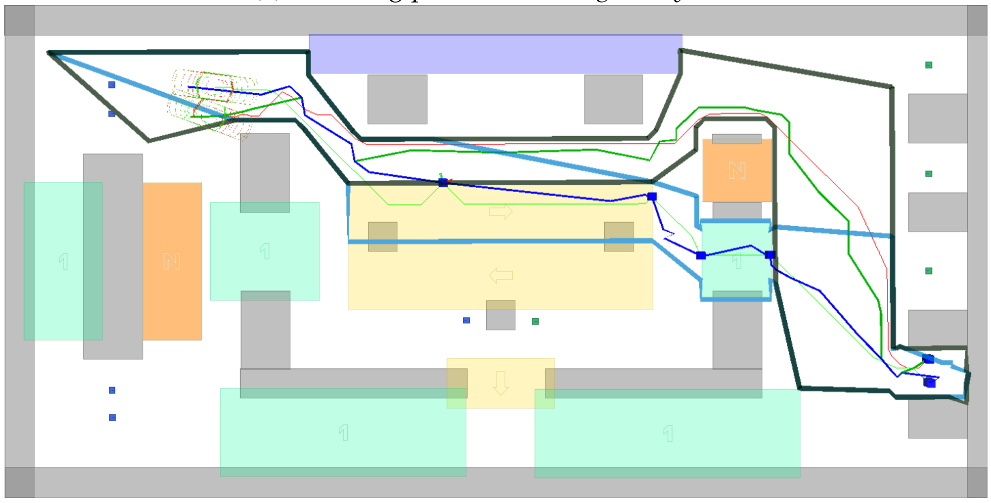


(b) Resulting high-level plan of the *New System*

Figure 7.10: Illustration of a use case for pursuing robots. Two robots are navigating consecutively a long corridor with two single robot zones. (a) shows the realization of this problem with the *Original System*; (b) visualizes the resulting planning windows and waypoints of the *New System*



(a) Resulting paths of the *Original System*



(b) Resulting high-level plan of the *New System*

Figure 7.11: Visualization of a goal execution of two robots for the same navigation goal. (a) depicts the mainly overlapping paths from the *Original System* (red, green) even through single robot zones; (b) shows the coordinated paths (blue, green) provided by the *New System*.

8 Conclusion

This chapter discusses and summarizes the work of this thesis. Additionally improvements of the system and future work are suggested.

8.1 Discussion

This thesis presented an approach for a hierarchical navigation system which is able to coordinate groups of autonomous logistics robots in industrial environments.
We introduced a formal description of this problem as a constrained satisfaction problem. This formulation results in a global optimization problem of finding the optimal plans for all robots in a given environment and a set of traffic regulating constraints. This is a general formulation which assumes that the information of the tasks of all robots is available beforehand. Since this information is not available we had to reduce this formulation to an optimization problem of an individual robot given the traffic regulating constraints and the current paths of all other robots.

In order to realize a solution of this problem we proposed concepts for generating an abstract environment representation and a high-level planning instance. Our environment representation brings up an expressive road map graph with an explicit representation of traffic regions and their constraints which is easily expendable. A new planning instance is used to generate a high-level plan on this road map graph. Therefore we enhanced an existing hierarchical planning approach with an additional planning layer. This layer, namely the

RegionPlanner, computes the high-level plan which is used and executed by the lower level planners. The *RegionPlanner* is intended to find fastest paths to a navigation goal considering all other robot paths. Therefore we adapted the A* algorithm and developed a heuristics able to consider all traffic regulating constraints. The additional planning layer operates decentralized on every robot, but uses the data of a central timetable to coordinate the common resource allocation. A high-level plan consists of waypoints, a restrictive planning window and a time schedule. This plan is communicated with the other robots via a centralized timetable.
The proposed algorithms allow the system to coordinate multiple robots with respect to different kind of traffic constraints and are intended to improve the overall performance of a multi-robot system.

We provided an extensive evaluation of the performance of the hierarchical navigation system. Therefore the planning instance and the multi-robot system had been investigated separately. The first evaluation regarded the computation time of the *RegionPlanner* considering different scenarios of increasing complexity. These evaluations showed that the developed heuristics and planner is able to cope with the different scenarios. Furthermore we suggested a worst case scenarios to challenge the worst case behavior of the planner.
The system evaluation showed a significant performance gain in comparison to the current implementation used especially when considering scenarios with higher number of allocatable traffic regions and multiple robots. We found that the new hierarchical navigation system is suggested to perform even better if more realistic scenarios with fixed goal stations are used.
For simpler scenarios with a few robots the system seems to be an overhead but it can still bring up a similar performance to the original system.

8.2 Future Work

The methods used for environment representation are able to generate an useful graph of the given environment. Nevertheless there are two aspects that could lead to further improvements.
In order to improve the path quality of our high-level plan we showed in the evaluation the effects of the implemented subdivision algorithm. The longest edge subdivision provided at least visually a better path quality but the performance was worsened dramatically because of the resulting graph structure. Hence there is a need of a method which can produce a better path quality, but still provides an acceptable performance. One algorithm supposed to achieve these requirements is the loop subdivision method discussed in Section 5.2.3 which has not been implemented within this work.
As a further improvement waiting positions in front of capacity regions could be integrated in the graph. Waiting positions could provide a well-regulated behavior of multiple robots waiting in front of an region entry. The current implementation may lead to a blocking behavior if multiple robots stand in front of an entrance and are waiting until the reserving robot exits the region. This should be basically a simple extension to the current graph generation step which is supposed to be implementable within the planning instances quite easily.

The implementation of the *RegionPlanner* is currently able to react on obstacles, for example a pallet or defect robot, which are not represented in the environment. A possible improvement is to share this information within our road map graph with other robots. The idea is simple, if one robot is not able to pass through a place in the environment, another robot would not be able too. An obstacle in the road map graph could be simply added by removing the corresponding edges from the graph.

The hierarchical navigation system uses currently a rather simple prioritization of robots. The first robot reserving a time slot on the central timetable receives the grant (first come first serve). This leads potentially to a globally not optimal task fulfillment by the fleet of robots. Therefore it would be probably an significant improvement if the prioritization for a fleet of robots in a logistics scenario takes into account the position and the order sequence of all robots.

Furthermore the navigation system has to be intensively tested and evaluated in practice to define strengths and weaknesses more accurate.

Bibliography

[1] D.D. Grossman. "Traffic control of multiple robot vehicles." In: *Robotics and Automation, IEEE Journal of* 4.5 (Oct. 1988), pp. 491–497. ISSN: 0882-4967 (cit. on p. 1).

[2] Y. Uny Cao, Alex S. Fukunaga, and Andrew Kahng. "Cooperative Mobile Robotics: Antecedents and Directions." In: *Auton. Robots* 4(1) (Mar. 1997), pp. 7–27. ISSN: 0929-5593 (cit. on p. 2).

[3] Howie M Choset. *Principles of robot motion: theory, algorithms, and implementation*. MIT press, 2005 (cit. on p. 9).

[4] Bruno Siciliano and Oussama Khatib. *Handbook of robotics*. Springer Science & Business Media, 2008 (cit. on p. 11).

[5] A. Kleiner, Dali Sun, and D. Meyer-Delius. "ARMO: Adaptive road map optimization for large robot teams." In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), 2011*. Sept. 2011, pp. 3276–3282 (cit. on p. 11).

[6] Alexander Kleiner, Bernhard Nebel, et al. "Behavior-based multi-robot collision avoidance." In: *IEEE International Conference on Robotics and Automation (ICRA), 2014*. IEEE. 2014, pp. 1668–1673 (cit. on pp. 12 sq.).

[7] Jing Wang and Suparerk Premvuti. "Distributed traffic regulation and control for multiple autonomous mobile robots operating in discrete space." In: *IEEE International Conference on Robotics and Automation, 1995. Proceedings., 1995*. Vol. 2. IEEE. 1995, pp. 1619–1624 (cit. on p. 13).

[8] Malcolm Ryan. "Constraint-based multi-robot path planning." In: *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. IEEE. 2010, pp. 922–928 (cit. on p. 13).

[9] Daniel Harabor and Adi Botea. "Hierarchical path planning for multi-size agents in heterogeneous environments." In: *IEEE Symposium On Computational Intelligence and Games, 2008.* IEEE. 2008, pp. 258–265 (cit. on p. 14).

[10] Joseph SB Mitchell and Christos H Papadimitriou. "The weighted region problem: finding shortest paths through a weighted planar subdivision." In: *Journal of the ACM (JACM)* 38.1 (1991), pp. 18–73 (cit. on pp. 15, 29).

[11] Douglas Demyen and Michael Buro. "Efficient triangulation-based pathfinding." In: *American Association for Artificial Intelligence.* Vol. 6. 2006, pp. 942–947 (cit. on p. 16).

[12] John Hershberger and Jack Snoeyink. "Computing minimum length paths of a given homotopy class." In: *Computational geometry* 4 (2) (1994), pp. 63–97 (cit. on p. 16).

[13] Joo Young Hwang et al. "A fast path planning by path graph optimization." In: *Systems, Man and Cybernetics, Part A: Systems and Humans, IEEE Transactions on* 33.1 (2003), pp. 121–129 (cit. on p. 16).

[14] Marcelo Kallmann. "Path planning in triangulations." In: *Proceedings of the IJCAI workshop on reasoning, representation, and learning in computer games.* 2005, pp. 49–54 (cit. on p. 16).

[15] Eitan Marder-Eppstein et al. "The Office Marathon." In: *IEEE International Conference on Robotics and Automation (ICRA)* (2010) (cit. on pp. 16, 20, 27, 32, 64).

[16] D. Fox, W. Burgard, and S. Thrun. "The dynamic window approach to collision avoidance." In: *Robotics Automation Magazine, IEEE* 4 (1).1 (Mar. 1997), pp. 23–33. ISSN: 1070-9932 (cit. on p. 17).

[17] Rachid Alami et al. "Multi-robot cooperation through incremental plan-merging." In: *Robotics and Automation, 1995. Proceedings., 1995 IEEE International Conference on.* Vol. 3. IEEE. 1995, pp. 2573–2579 (cit. on p. 17).

[18] R. Chatila et al. "Integrated planning and execution control of autonomous robot actions." In: *Robotics and Automation, 1992. Proceedings., 1992 IEEE International Conference on*. May 1992, 2689–2696 vol.3 (cit. on p. 18).

[19] Morgan Quigley et al. "ROS: an open-source Robot Operating System." In: *ICRA workshop on open source software*. Vol. 3. 3.2. 2009, p. 5 (cit. on p. 19).

[20] Botsch Steinberg Bischoff et al. "OpenMesh–a generic and efficient polygon mesh data structure." In: *In OpenSG Symposium*. 2002 (cit. on p. 23).

[21] James F Allen. "Maintaining knowledge about temporal intervals." In: *Communications of the ACM* 26.11 (1983), pp. 832–843 (cit. on p. 24).

[22] S. M. LaValle. *Planning Algorithms*. Cambridge, U.K.: Cambridge University Press, 2006 (cit. on pp. 31, 38 sq.).

[23] L. Paul Chew. "Constrained delaunay triangulations." In: *Algorithmica* 4.1 (), pp. 97–108. ISSN: 1432-0541 (cit. on p. 34).

[24] Charles Loop. "Smooth subdivision surfaces based on triangles." MA thesis. Dept. of Mathematics, University of Utah, Aug. 1987 (cit. on p. 44).

[25] Peter E Hart, Nils J Nilsson, and Bertram Raphael. "A formal basis for the heuristic determination of minimum cost paths." In: *Systems Science and Cybernetics, IEEE Transactions on* 4.2 (1968), pp. 100–107 (cit. on pp. 45, 48).

[26] Robin Murphy. *Introduction to AI robotics*. MIT press, 2000 (cit. on p. 45).

[27] Alex Nash et al. "Theta^*: Any-Angle Path Planning on Grids." In: *Proceedings of the National Conference on Artificial Intelligence*. Vol. 22. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2007, p. 1177 (cit. on pp. 48, 50).

[28] Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated planning: theory & practice*. Elsevier, 2004 (cit. on p. 49).

[29] Antonin Guttman. *R-trees: a dynamic index structure for spatial searching*. Vol. 14. 2. ACM, 1984 (cit. on p. 72).