Decentralised Multi-Robot Systems

Towards Coordination in Real World Settings

Thesis submitted in accordance with the requirements of
the University of Liverpool for the degree of Doctor in Philosophy by

Daniel Claes

April 2018

Contents

List of Figures

List of Tables

Preface

This thesis is primarily my own work. The sources of other materials are identified.

Abstract

In recent years, Multi-Robot Systems (MRS) have gained significant interest in research and in industry (Khandelwal and Stone, 2017; E. Schneider et al., 2016; Amato et al., 2015; Alonso-Mora et al., 2015b; Enright and Wurman, 2011). Manufacturers are moving away from large one-size-fits-all productions to more customisable on demand production, which result in smaller and smaller batch sizes. Additionally, in order to be able to increase productivity even further, more and more tasks in the production process have to be automated. To accommodate these changes, industry is facing major shifts in how the products are produced and in particular the role robotic platforms are playing.

Previously, robots have mainly been used in a static manner, i.e. performing a singular repetitive task over and over again with high precision and speed. When multiple robots are employed in such a setup, each robot performs a dedicated task, with no interaction with the other robots.

While this approach was suitable for large-scale productions, it cannot maintain the same productivity for highly customisable products. Additionally, many tasks in the production process require that the robots are mobile, since they are spatially distributed. One example is for instance retrieving items from different locations in a warehouse. Furthermore, another requirement is that every robot should be able to handle many different tasks and more importantly, many robots should work together in a team towards a common goal.

These new requirements introduce various new challenges. As an example, since the robots are mobile, they should be able to perform the tasks alongside the human workers. Likewise, since multiple robots have to work together, a new challenge is to coordinate such MRS.

The work presented in this thesis focuses on the core issues when deploying MRS in the physical world. We focus on the task of warehouse commissioning as a running example. The environment for this task is highly dynamic, adaptive and complex, since new orders can appear at any time and priorities might change. A major issue is to coordinate the robots, while taking current and possible future tasks into account.

One solution is a *centralised* planning entity, which knows about all tasks and robots in the team and assigns the tasks accordingly. While in the case of a handful robots, a good assignment can usually be calculated in a straight forward manner, a problem with a centralised system arises when more and more robots are added to the system. The

number of possible assignments rises exponentially with every additional robot. Thus, planning times increase and it might become infeasible to provide an optimal plan in time or to respond quickly to changes.

On the other hand, in a *decentralised* solution, each robot decides on its own. Thus, it accumulates all necessary information, and calculates a plan based on this information. While the robots might not have all information available, this is in many cases not necessary. The planning robot is mainly interested in its *own* actions. While the robot should take the other robots into account, this effect can be approximated, and not every single action of the other robots is needed. This results in a much less complex planning problem, which allows the robot to re-plan *online*, as soon as the environment changes.

In this thesis, we focus on such decentralised solutions for MRS that can run online on the robots. We investigate navigation, decision making and planning algorithms that are suitable for problems in which the tasks are highly dynamic and spatially distributed, such as the warehouse commissioning example. We explore how a team of robots can navigate safely in a shared environment with humans. We apply Monte Carlo sampling techniques and trajectory rollouts as used in the commonly used Dynamic Window Approach (DWA) (Fox et al., 1997), while taking the localisation uncertainty into account. We show that our resulting navigation method is robust and able to run decentralised on the robots.

To facilitate formal evaluation of planning and decision making algorithms, a formal framework called Spatial Task Allocation Problems (SPATAPs) is introduced, that enables us to capture and analyse these problems in the well known Markov Decision Process (MDP) (Puterman, 1994) and Multi-Agent Markov Decision Process (MMDP) (Boutilier, 1996) frameworks. The commonly used MDP solution methods, i.e. value iteration and dynamic programming, fail to provide a solution, due to the large problem space. We investigate whether we can exploit the structure of these problems and introduce approximations to enable planning using the common solution methods. We further refine the framework to formally capture the warehouse commissioning task. A solution method based on Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006) is introduced, using computationally cheap greedy roll-out strategies. We show that the resulting approach can yield significantly higher performance than previous approaches, while still being able to plan within the magnitude of seconds, which allows for online re-planning on the robots.

Finally, the decision making algorithm and the navigation approach are combined in a proof-of-concept application, in which three youBots are used in a physical warehouse commissioning setup.

Acknowledgements

While this section comes early in the thesis, it is the last one to write. After almost five years, my research as PhD-student is concluded. I would not have made it through this interesting but also challenging time without the support, help and guidance of the people around me.

First of all, I would like to thank you, Karl, for your support as my supervisor and your guidance. Without you, I would not have been able to complete this work. You allowed me to do this work with all the freedom I could wish for, with the gentle reminders of some upcoming deadlines. You enabled my frequent weekends to visit Renée and also a longer visit in Delft. Thank you for all that you have done!

Joscha and Basti, we lived together for the last years and all moved together to Liverpool to do our PhD there. Together with Karl, Daan and Frans, we came from Maastricht University to invade Liverpool and build up the smARTLab. First, we selected all the best tools for the workshop at university, and after a while we had built up our own workshop, with equal or even better tools in our living room. We spent some days and nights on completely different rhythms, the one going to bed, when the others woke up, but it was always nice to know that someone was home and could talk about the latest Chinese gadgets.

Thanks to you, Richard, I learned a lot about 3D printing, and with Basti and Joscha, we spent countless hours on RoboCup and RoCKIn projects, and besides we became the local amateur FPV drone racing club.

With you, Eric and Basti, I also found nice bouldering partners. We spend many nights at the climbing hangar to try our best at that "stupid red", "slopy white" or sometimes even "crimpy yellow" routes.

Another thanks goes to my collaborators, and especially to you, Frans and Daniel. You gave valuable insights, feedback comments and ideas, which helped me to push my research further in one or in another directions.

I would like to express my thanks to the people at the Magazino GmbH, who allowed me to get a glimpse of the actual *real world*, not the lab-fabricated so-called *real-world settings*. Thank you for the confidence that I could test and implement some of my ideas for the order batching of the Toru robots.

A special thanks is to Matthijs, which read through early versions of the thesis and gave me valuable feedback. We spent many outdoor vacations together, and hopefully,

now that I am also living in Eindhoven, the previously rare meetings can become more regularly.

Another very important support has always been my family, you have always been there if I need you and you are not mad if you do not hear anything from me for a couple of weeks. You have been very understanding and supportive in stressful periods and it is always nice to come home.

Als laatste wil ik nog mijn aanstaande vrouw bedanken. Renée, we hebben samen al heel veel proeven doorstaan. De periode van mijn PhD was zeker ook geen makkelijke tijd voor ons. Na een jaar ging ik naar Liverpool en jij naar Den Haag. Dat betekende veel vliegen en we konden ons meestal alleen in het weekend zien. Maar gelukkig hebben wij ook deze tijd goed doorstaan. Zonder jouw steun en de vele leuke dingen die wij samen hebben mee gemaakt was het mij waarschijnlijk ook niet gelukt om deze scriptie af te krijgen. De reizen naar de Alpen, Oman, Marokko en diverse andere bestemmingen zijn altijd leuke momente om op terug te kijken. Er komt nu een nieuwe periode op ons af, in toekomst als man en vrouw. Ik ben blij dat ik dit met jou mag ervaren. Ik hou van jou!

Thank you all!

Chapter 1

Introduction

Traditionally, Economies of Scale (Baye and Beil, 2006), i.e. reducing the costs per item by larger batches, has driven the progress in manufacturing. For instance, Ford factories were the first to use automation and industrialisation in the production to achieve a large batch-size of a single product - the Ford Model T, available only in a single version. This car was so popular, it was the only car which was affordable on a normal income, that during the mid twenties more than half of the world's cars were nearly identical Ford Model T's (Ford, 2013).

The trend of automation has continued. Especially in the car industry, more and more robots are employed to perform repetitive tasks with high speed and accuracy. Even complete production lines are being automated (Wired, 2014).

With the growing customisation demands from the population (Flynn and Vencat, 2012), the manufacturers are moving away from the large one-size-fits-all productions and try to accommodate the specific wishes of the individual customers.

However, to keep the same level of productivity while increasing the diversity of the products and lowering the batch sizes, more and more tasks have to be automated. Since most of the repetitive tasks are already being performed by robots, the challenge is to automate the more complex and flexible tasks, which are still being performed by humans. These tasks typically require moving around and changing environments, for instance, fetching items from a warehouse or tasks which require multiple people working together.

The previous approach of statically programmed robots is not able to handle these complex tasks. The robots are programmed by experts to perform a specific task, and as soon this task or the environment changes, even only slightly, the programming has to be changed. This programming takes a long time and is difficult to perform (Pan et al., 2012). Thus, this often entails high costs for specialised human personnel that needs to interact with the robots to adjust their programming for these changes.

Additionally, since the robots follow fixed routines, the robots will perform the task no matter the circumstances. This can lead to dangerous situations if the environment is not as expected. As a result, there are high safety constraints and the robots have to operate inside cages, with safety controllers that stop the robot as soon as a human enters

the working space (International Organization for Standardization, 2011a; International Organization for Standardization, 2011b). Unfortunately, even with those precautions, major accidents have occurred in the past (Associated Press, 2015).

This creates a need for adaptive, flexible Multi-Robot Systems (MRS) that can react to changes, without the need of reprogramming and work together and alongside humans in the same environment. These requirements imply that the robots have to be mobile and capable of performing multiple different tasks.

This trend in production has often been referred to as the fourth industrial revolution, or *Industry 4.0*, which was coined at the Hanover Fair in 2011 (Kagermann et al., 2011).

The goal is to be able to produce customised items in smaller production batches, with the ultimate goal to be able to economically and efficiently manufacture highly customised products in "Batch Size 1" (Hannover Messe, 2015) while maintaining the same or even higher level of productivity.

The main technology which will enable high productivity with customisation are flexible and cooperative mobile MRS. However, there are many open challenges which need to be resolved. An important challenge is, how (multiple) robots can share and navigate in the workspace safely while humans are present. Furthermore, how can a team of robots be coordinated in an efficient way, such that it can react quickly on changes in the environment, such as new tasks or changing priorities?

Industry, especially in high wage countries, can only remain competitive in future by addressing these challenges. In this thesis, we tackle those challenges and present algorithms and solution methods to enable adaptive MRS that are deployable in the physical world.

First, we investigate how a team of robots can navigate safely in a shared environment with humans. We use the idea of the Velocity Obstacle (VO) paradigm and extend it to include localisation uncertainty. We investigate how we can tune the behaviour of the robots using various cost functions to enable safe navigation within the presence of humans. We explore Monte Carlo sampling techniques and trajectory rollouts as used in the commonly used Dynamic Window Approach (DWA), leading to a robust navigation method that is running decentralised on the robots.

Second, we focus on the decision making and planning for problems in which the tasks a spatially distributed. A formal framework called Spatial Task Allocation Problems (SPATAPs) is introduced, that enables us to capture and analyse these problems in the well known Multi-Agent Markov Decision Process (MMDP) framework. Unfortunately, the commonly used MMDP solution methods, i.e. value iteration and dynamic programming, fail to provide a solution, since the state and/or action space is too large. We show how we can exploit the structure of these problems and introduce approximations to enable planning using the common solution methods.

We further refine the framework to capture the warehouse commissioning task. A solution method based on Monte Carlo Tree Search (MCTS) is introduced, using computationally cheap greedy roll-out strategies. The resulting decision making algorithm

is evaluated against the previous value iteration based approach. We show that it can yield significantly higher performance while still being able to plan within the magnitude of seconds, which allows for online re-planning on the robots.

Finally, the decision making algorithm and the navigation approach are combined for a proof-of-concept application, in which up to three youBots are used in a real world warehouse commissioning setup.

1.1 Content of this Thesis

This thesis focuses on two core issues when employing MRS in the real world, namely navigation and coordination. More specifically, we investigate how to navigate safely in a shared workspace, under the presence of other robots, humans, and previously known and also unmapped static obstacles. Additionally, we will look into how to the robots can decide which action to take, while considering the robots in the system. In both cases, we are aiming for a system that runs *decentralised* and *online* on the robots.

To describe the content of this thesis in more detail, we will use warehouse commissioning as the running example. This task is highly complex, since a team of robots has to commission items from a warehouse, while considering the current and possibly future orders. The orders contain one or multiple items that have to be picked, and brought back to the depot, where it can be packed and sent to the customer.

There are different approaches for planning for such a MRS, which can generally be categorised in *centralised* or *decentralised* systems. Additionally, the planning can be done *offline* or *online*. This means that the plans are either calculated before the run (i.e. offline) and afterwards followed during execution, or that the planning is performed during the execution of the run, thus, online.

In a *centralised* planning system the controlling entity is planning for all robots. In our example, that would mean that for instance the warehouse management software would assign specific tasks to the robots, which then only follow and execute these predefined plans. These approaches often plan offline, with high processing power and long planning times.

While these systems are shown to work very well, since the warehouse management software has access to all information, it is also a notoriously difficult problem to solve in a good way. For each additional robot, we add exponentially more possibilities to assign the tasks. This leads to longer and longer planning times, especially if the number of orders and number of robots are high. In many cases these systems follow simple rules in order to be able to plan at all. Another problem is that the planning time is high. This means that the system cannot react quickly to changes, since it needs to re-plan for the entire team. Likewise, if this planning entity has a failure, the whole system breaks down and all the robots do not get a new plan. Furthermore, with more and more sensors being implemented and more data becoming available the complexity of the planning problem increases immensely.

On the other hand, in a *decentralised* system, the robots plan individually. Thus, they gather information, which in our example could be broadcasts from the other robots about their positions and the currently active orders, and then the robots try to come up with their next best action based on the current information. An action could be for instance driving to a new position, picking up an item or unloading the items. One of the main difficulties is that for determining a good next action, the robots have to predict what the other robots will most probably do, which we will investigate in this thesis. Since planning for only one robot is less complex, it can run *online*, i.e. it can be recalculated many times. For instance, if a new high priority order comes in, the system will immediately react to this. Another advantage of this approach that robots can be added and removed to the system "on the fly". For instance, if a robot breaks down, it does not broadcast its position anymore and as a result the other robots will automatically plan without it.

The general principles of the work presented in this thesis can be applied to many domains in which teams of robots have to perform tasks.

The following research questions have been derived:

1. To what extent can we ensure decentralised collision free navigation in a highly dynamic setting considering that other robots, humans and other known and unknown obstacles might be present? *Chapter 3*

2. To what extent can we derive a formal framework within which Multi-Robot coordination can be described, understood and evaluated? *Chapter 4*

3. To what extent can this formal framework be used to implement and deploy decentralised Multi-Robot Systems? *Chapter 5*

4. To what extent can the developed framework and algorithms be deployed in an Industry 4.0 context, considering single robots, multiple robots and humans? *Chapters 3 and 6*

1.2 Structure and Contributions of this Thesis

In the following, we will overview the structure of this thesis and provide a brief summary of the contributions made in each chapter:

- Chapter 2 introduces the necessary theoretical background and foundations for this work. We introduce the Markov Decision Process (MDP)-framework and its variations and present an approach for navigation with multiple robots. Additionally, the robotic platforms and core software components are introduced. The related work will be discussed in each chapter separately.

- In Chapter 3, we extend earlier work on Multi-Robot collision avoidance (Claes et al., 2012; Hennes et al., 2012) and introduce a way to include humans into

the system. Additionally, we show how the approach can be combined with a commonly used navigation methods, the DWA, and evaluate our approach with up to 10 robots in simulation and up to three robots and one human in real world experiments. This chapter is based on:

  - Claes, D., Hennes, D., and Tuyls, K. "Towards Human-Safe Navigation with Pro-Active Collision Avoidance in a Shared Workspace". In: *Proceedings of the IROS Workshop on On-line decision-making in multi-robot coordination DEMUR)*. 2015.
  - Claes, D., and Tuyls, K. "Multi robot collision avoidance in a shared workspace". In: *Autonomous Robots special Issue on Distributed Robots* (under submission).

- Chapter 4, we define a formal framework for dealing with decentralised Multi-Robot coordination problems, which we call SPATAPs. Within this framework, we show its applications and possible solution methods in an example of a *dirt-cleaning grid world*. Our evaluation shows that our method is able to handle the complex environment yielding a good performance when comparing against optimal (where possible) and another state-of-the-art partitioning approach. This chapter is based on:

  - Claes, D., Oliehoek, F., Baier, H., and Tuyls, K. "Decentralised Online Planning for Multi-Robot Warehouse Commissioning". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems. International Foundation for Autonomous Agents and Multiagent Systems.* 2017, pp. 492–500.

- Chapter 5 builds upon the previously introduced SPATAP-framework and extends the previous example of a dirt world to a Warehouse Commissioning environment. We present how MCTS can be applied in the SPATAP-framework by including computationally cheap greedy policies, which take their inspiration in auctioning and partitioning. Our evaluation demonstrates the good performance of the system, while achieving planning times in the magnitude of seconds. This chapter is based on:

  - Claes, D., Robbel, P., Oliehoek, F. A., Tuyls, K., Hennes, D., and Hoek, W. van der. "Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems. International Foundation for Autonomous Agents and Multi- agent Systems.* 2015, pp. 881–890.

- In Chapter 6, we combine the collision avoidance algorithm from Chapter 3 with the decision making approach from the previous chapter in order to implement a prototype of a real world warehouse commissioning system with multiple robots. We introduce the additionally necessary components for grasping, namely object recognition and inverse kinematics for the mobile manipulator and present the system, which is evaluated with two and three robots.

- Finally, in Chapter 7, we conclude this thesis and present possible directions for future work.

Chapter 2

Background

The work in this thesis extends and combines various approaches to achieve the goal of flexible decentralised Multi-Robot Systems (MRS), with the focus on the navigation and the decision making in the multi-robot coordination problem. In this chapter, we will introduce necessary theoretical background for the approaches, which will act as basis for the remainder of the work. Note that this will give the reader an introduction to the used frameworks, if the reader is familiar with these, the respective sections can be skipped. The related work that is specific for each chapter will be discussed in the respective chapter.

First, in Section 2.1, we will discuss, why flexible navigation is necessary and which are the shortcomings of current solutions. We will introduce the Velocity Obstacle (VO) paradigm (Fiorini and Shiller, 1998), with its extensions and show how dynamic and kinematic constraints of the robots can be taken into account within the VO framework. Afterwards, we introduce three different possibilities on how to select a collision-free velocity.

In the second part of the chapter (Section 2.2), we will discuss the decision theoretical frameworks that lay the foundation of this work. The Markov Decision Process (MDP) framework (Bellman, 1957a) will be introduced with possible solution methods and we look into the extensions for multiple robots. We introduce Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006), which can be used as an heuristic methods to plan in MDPs.

Finally, in Section 2.3, we will introduce the two robot platforms that are used in the practical experiments of this work and the main software components.

2.1 Navigation in a Shared Workspace

While some people see navigation in a two dimensional environment as a solved problem, this usually assumes a *static world* such as the commonly used Dynamic Window Approach (DWA) (Fox et al., 1997). In order to work well, the DWA requires that all obstacles can be detected and do not move during execution. This assumptions does not hold in a shared workspace environment. There are humans and other robots that

move around, and we can assume that everybody aims to avoid collisions to a certain extent. This leads to a highly dynamic environment, in which it is not easy to navigate.

In some other approaches, the robots are centrally coordinated, and kept separated from the humans. However, this approach is only feasible in settings where the central controller can communicate with all robots and there are strict rules where and when the robots and humans are allowed to move around. Additionally, when the centralised controller fails, the whole system breaks down. An example of such a system is KIVA (Wurman et al., 2008; Enright and Wurman, 2011).

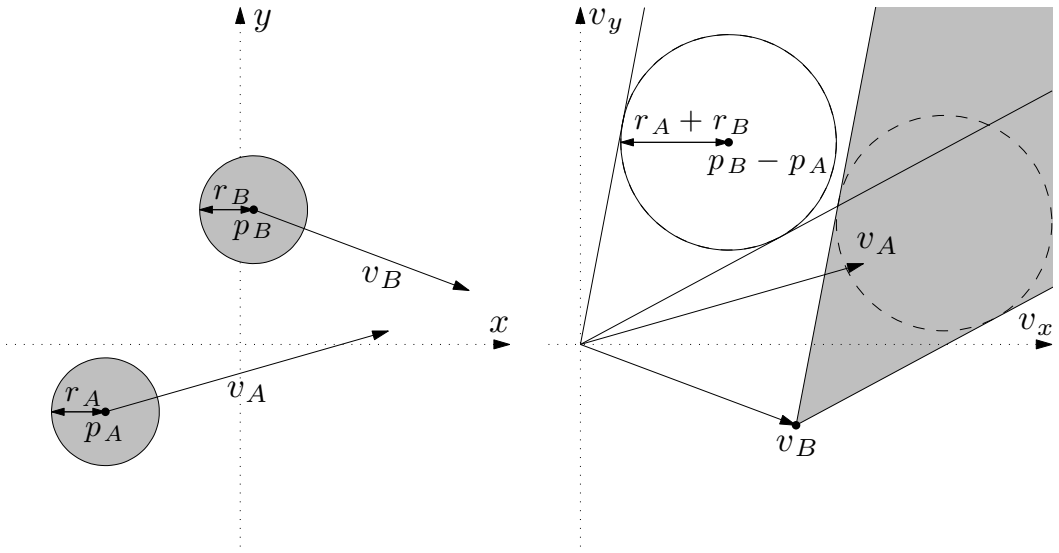
Another downside of these centralised approaches is that the environment has to be built according to the specifications of the system. Which means that, for instance in the KIVA case, the complete warehouse needs to be constructed such that there is a specific layout in which the system works, e.g. there is enough space for the robots to move. This is not always feasible, since to completely rebuild the work environment is too costly. Hence, there is the need for flexible navigation solution in existing and dynamic environments. In the following, we will discuss the underlying principles for such an approach.

2.1.1 The Velocity Obstacle Paradigm

This section describes the VO paradigm as introduced in (Fiorini and Shiller, 1998). First, we will present the construction of the various types of the VOs that have evolved over time to take reciprocity into account. Afterwards, three examples of how to select a new collision free velocity are explained and how dynamic and movement constraints for different type of robots can be taken into account.

The VO was introduced as an approach to deal with dynamic, i.e. moving, obstacles. These could be for instance other robots. The VO is the geometric representation of all possible velocities that will eventually result in a collision, in the velocity space of the planning robot. The assumption is that the dynamic obstacle maintains the observed velocity. To cover speed changes of the dynamic obstacles, it is necessary that the controller runs multiple times per second. This results in a piece-wise linear approximation of the problem.

In other words, the approach translates the problem into the velocity space, more specifically, the space of all possible velocity vectors in x (forward and backwards) and y (left and right) direction from the point of view of the planning robot. In this space, we calculate the areas that will lead to collision at some point in the future under the assumption that every velocity remains constant. The VO paradigm assumes that the robots are able to instantaneously accelerate to any velocity in the two dimensional velocity space, which is only feasible for a so-called holonomic robot. In Section 2.1.2 we will show how different kinematic and dynamic constraints can be incorporated in the model.



(a) Workspace configuration (b) VO

FIGURE 2.1: Creating the different velocity obstacles out of a workspace configuration. (a) A workspace configuration with two robots R_A and R_B. (b) Translating the situation into the velocity space and the resulting VO for R_A.

The subsequent definition of the VO assumes planar motions, though the concept extends to 3D motions in a straight forward manner as shown in (Alonso-Mora et al., 2015b).

Let us assume a workspace configuration with two robots (R_A and R_B), shown in Figure 2.1a, driving towards their goals with their velocities (v_A and v_B). While it might not be directly evident from that picture, we can show that the two robots are on collision course, when they would maintain their current speeds. In order to show that, we translate the situation into the *velocity space* of robot R_A. Where the *velocity space*, is the space of linear, i.e. x- and y-directional, velocities that R_A can choose from.

If the position and speed of the other robot R_B is known to R_A, we can mark a region in the velocity space which leads to a collision under current velocities and is thus unsafe. This region resembles a cone with the apex at R_B's velocity v_B, and two rays that are tangential to the convex hull of the Minkowski sum of the footprints of the two robots. The Minkowski sum for two sets of points A and B is defined as:

$$\mathcal{M}_{A,B} = \{a + b \mid a \in A, b \in B\}. \tag{2.1}$$

For the remainder of this thesis, we define the $\oplus$ operator to denote the convex hull of the Minkowski sum such that $A \oplus B$ results in the points on the convex hull of the Minkowski sum of A and B.

The direction of the left and right ray is then defined as:

$$\theta_{left} = \max_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel}), \tag{2.2}$$

$$\theta_{right} = \min_{p_i \in \mathcal{F}_A \oplus \mathcal{F}_B} atan2((p_{rel} + p_i)^{\perp} \cdot p_{rel}, (p_{rel} + p_i) \cdot p_{rel}), \tag{2.3}$$

where p_{rel} is the relative position of the two robots and $\mathcal{F}_A \oplus \mathcal{F}_B$ is the convex hull of the Minkowski sum of the footprints of the two robots. The $a^{\perp}$ returns the perpendicular vector to a in counter-clockwise direction as defined in (Hill Jr, 1994). The $atan2$ expression computes the signed angle between two vectors. The resulting angles θ_{left} and θ_{right} are left and right of p_{rel}. If the robots are disc-shaped, the rays are the tangents to the disc with the radius $r_A + r_B$ at centre p_{rel} as shown in Figure 2.1b. The angle can then be calculated as:

$$\theta_{left} = -\theta_{right} = arcsin(\frac{r_A + r_B}{|p_{rel}|}). \tag{2.4}$$

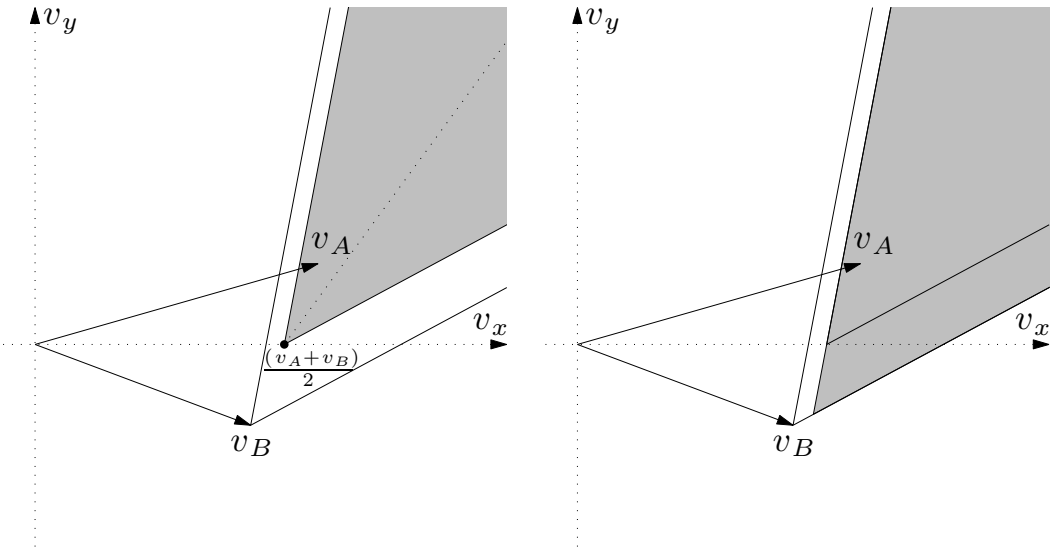
In our example, in Figure 2.1b, it can be seen that robot R_A's current velocity vector v_A points into the VO, thus we know that R_A and R_B are on collision course. As a result, the robot should adapt its velocity in order to avoid collision.

Each agent computes a VO for each of the other agents, in our example R_B also calculates the VO induced by R_A. If all agents at any given time-step adapt their velocities such that they are outside of all VOs, the trajectories are guaranteed to be collision free.

However, oscillations can still occur when the robots are on collision course. For example, all robots could select a new velocity outside of all VOs independently, hence, at the next time-step, the old velocities pointing towards the goal will become available again. Thus, the robots would select their old velocities, which will be on collision course again for the next calculation, where each robot selects again a collision free velocity outside of all VOs.

To overcome these oscillations, the Reciprocal Velocity Obstacle (RVO) was introduced in (Berg et al., 2008). The surrounding moving obstacles are in fact also pro-active agents and thus aim to avoid collisions too. Assuming that each robot takes care of half of the collision avoidance, the apex of the VO can be translated to $\frac{v_A + v_B}{2}$ as shown in Figure 2.2a. This leads to the property that if every robot chooses the velocity outside of the RVO which is closest to the current velocity, the robots will avoid to the same side. However, in some situations the robots will not avoid to the same side, since the selected velocity should also make progress towards its goal location, and therefore the closest velocity which is collision free is on the *wrong* side of the RVO.

To counter these situations, the Hybrid Reciprocal Velocity Obstacle (HRVO) was introduced in (Snape et al., 2009; Snape et al., 2011). Figure 2.2b shows the construction of an HRVO. To encourage the selection of a velocity towards the preferred side, e.g. left in this example, the opposite leg of the RVO is substituted with the corresponding leg of the VO. The new apex is the intersection of the line of the one leg from RVO and the line of the other leg from the VO. This reduces the chance of selecting a velocity on the *wrong* side of the velocity obstacle and thus the chance of a reciprocal dance, while not over-constraining the velocity space. The robot might still try to pass on the *wrong*



(a) RVO (b) HRVO

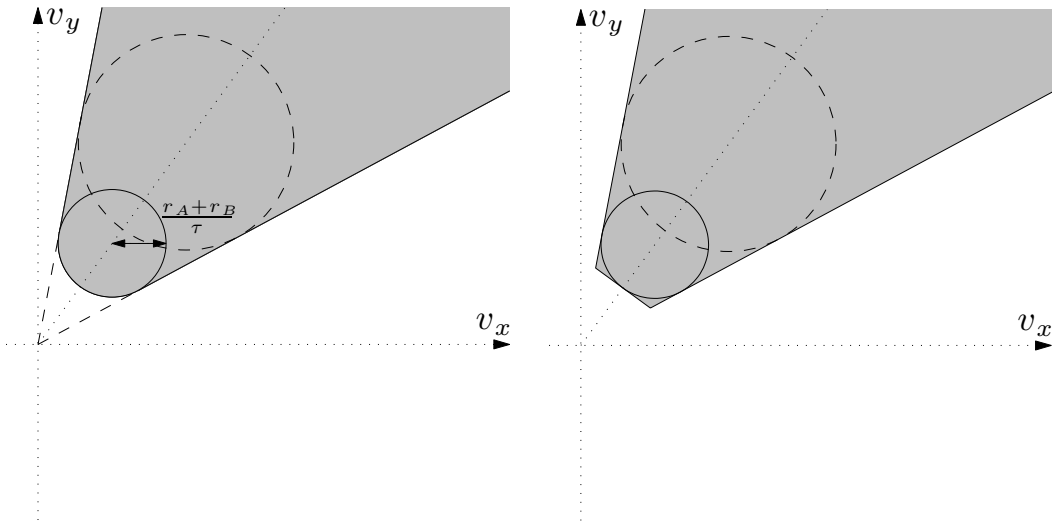
FIGURE 2.2: (a) Translating the VO by $\frac{v_A+v_B}{2}$ results in the Reciprocal Velocity Obstacle (RVO), i.e. each robot has to take care of half of the collision avoidance. (b) Translating the apex of the RVO to the intersection of the closest leg of the RVO to the own velocity, and the leg of the VO that corresponds to the leg that is furthest away from the own velocity. This encourages passing the robot on a preferred side, i.e. in this example passing on the left. The resulting cone is the Hybrid Reciprocal Velocity Obstacle (HRVO).

side, e.g. another robot induces a HRVO that blocks the whole side, but then soon all other robots will adapt to the new side too.

Another problem occurs when the workspace is cluttered with many robots and these robots to not move or to only move slowly. As shown Figure 2.1b, the VOs are translated by the velocity of the other agents. Thus, in these cases, the apexes of the VOs are close to the origin in velocity space. Additionally, if static obstacles such as walls are included, any velocity will lead to a collision eventually, thus rendering the robots immobile. This problem can be solved using truncation.

The idea of truncating a VO can best be explained by imagining a static obstacle. Driving with any velocity in the direction of the obstacle will eventually lead to collision, but not directly. Hence, we can define an area in the velocity space, for which the selected velocities are safe for at least τ time-steps. The truncation has then the shape of the Minkowski sum of the two footprints shrunk by the factor τ. If the footprints are discs, the shrunken disc that still fits in the truncated cone has a radius of $\frac{r_A+r_B}{\tau}$, see Figure 2.3a. VO^τ denotes a truncated velocity obstacle. The truncation can be closely approximated by a line perpendicular to the relative position and tangential to the shrunken disk as shown in Figure 2.3b. This enables easier calculations, since then each VO is defined by one line segments and two rays.

Applying the same method as creating a HRVO and RVO from a VO, we can create a truncated HRVO and truncated RVO ($HRVO^\tau$ and RVO^τ, respectively) from VO^τ



(a) Truncated VO (VO^{τ}) (b) Approximated VO^{τ}

FIGURE 2.3: Truncation. (a) Truncation of a VO of a static obstacle at $\tau = 2$. (b) Approximating the truncation by a line for easier calculation.

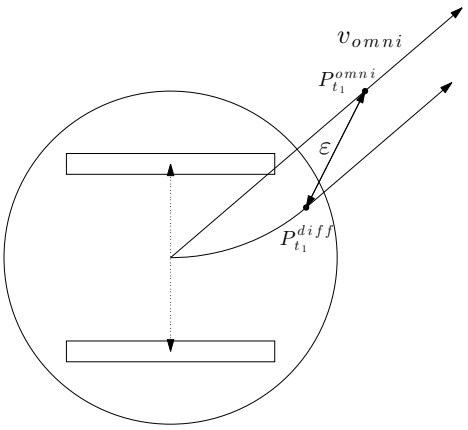


FIGURE 2.4: The tracking error (ε) is defined as the difference between the position that a holonomic robot would be in after driving with v_{omni} for t_1 ($P_{t_1}^{omni}$) and the position of the differential drive robot at t_1 ($P_{t_1}^{diff}$).

by translating the apex accordingly.

2.1.2 Incorporating Kinematic and Dynamic Constraints

As previously mentioned, the VO paradigm assumes that the robots are able to instantaneously accelerate to any velocity in the two dimensional velocity space. This implies that the velocity obstacle approach requires a fully actuated *holonomic* robot, which means that it is able to accelerate into any direction from any state. However, differential drive robots with only two motorised wheels are much more common due to their lower cost. Additionally, all robots can only accelerate and decelerate within certain

dynamic constraints. In this section, we will show how to incorporate these dynamic and kinematic constraints into the VO framework.

For a holonomic robot, when the acceleration limits and motion model are known, the region of admissible velocities can be calculated and approximated by a convex polygon. This region defines the set of velocities that is currently achievable. In other words, we limit the allowed velocity space by calculating the maximal and minimally achievable velocities in both x and y direction, and only allow velocities inside this region.

A method to handle non-holonomic robot kinematics has been introduced in (Alonso-Mora et al., 2010). The approach to handle dynamic and kinematic constraints can be applied to any VO-based approach. The underlying idea is that any robot can track a holonomic speed vector, i.e. the vector a holonomic robot would drive, with a certain tracking error ε. We refer to this vector as holonomic velocity as in (Alonso-Mora et al., 2010).

This tracking error depends on the direction and length of the holonomic velocity, for example, a differential drive robot can drive an arc and then along a straight line which is parallel to the holonomic speed vector in that direction as shown in Figure 2.4. The time needed to get parallel to the holonomic trajectory is defined as t_1. The tracking error (ε) is then defined as the difference between the position that a holonomic robot would be in after driving with a holonomic speed vector (v_{omni}) for t_1, shown as ($P_{t_1}^{omni}$), and the position of the differential drive robot at that time ($P_{t_1}^{diff}$).

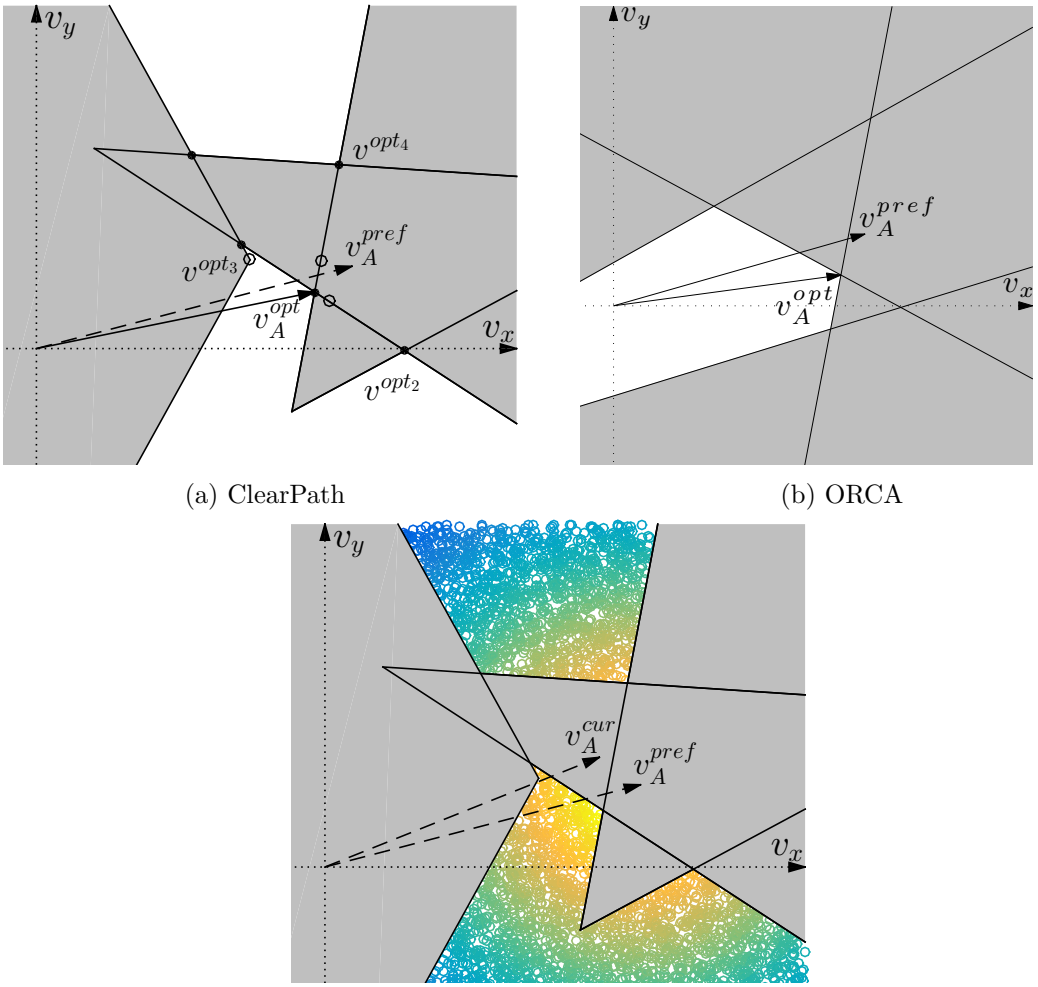
A set of allowed holonomic velocities is calculated based on the current speed and a maximum tracking error ε. To allow smooth and collision free navigation, the virtual robot footprints have to be increased by the tracking error, ε, since the robots only track the desired holonomic velocity with the defined error.

Using this approach, we can approximate the region of possible holomonic velocities using a polygon, and only allow the robots to choose a velocity within that region. In the next section, we will introduce three possible methods to select a new collision-free velocity.

2.1.3 Selection of a Collision-Free Velocity

When all velocity obstacles are calculated, the union of these velocity obstacles depicts the set of velocities that will eventually lead to a collision. Vice versa, the complementary region is the region that holds all *safe velocities*, i.e. velocities that are collision-free. If we are using truncation, the region is collision free for at least the defined τ time-steps. Additionally, we limit the velocity space according to the dynamic and kinematic constraints, as explained in the previous section.

The new velocity has to be selected within the remaining region. In order to do this efficiently, there are several ways to calculate the new velocity. Usually, we are following a global plan, which gives us a general direction in which we want to move. This is our preferred velocity v^{pref}. In the past, some algorithms were introduced that aim to solve this problem efficiently, namely the ClearPath algorithm (Guy et al., 2009) and



(a) ClearPath

(b) ORCA

(c) Monte Carlo Sampling

FIGURE 2.5: (a) ClearPath enumerates intersection points for all pairs of VOs (solid dots). In addition the preferred velocity v_A is projected on the closest leg of each VO (open dots). The point closest to the preferred velocity (dashed line) and outside of all VOs is selected as new velocity (solid line). The next best points are shown for reference. (b) Optimal Reciprocal Collision Avoidance (ORCA) creates a convex representation of the *safe velocity space* and uses linear programming to find the closest point to the preferred velocity. (c) We can also use Monte Carlo sampling to select the best velocity. The distance for each sample to the preferred velocity (dashed line) is evaluated. If the sample falls within any VO, it is discarded. Yellow shows a high rating and blue is a low rating.

ORCA (Berg et al., 2011). ClearPath follows the general idea that the collision free velocity that is closest to the preferred velocity is: (a) on the intersection of two line segments of any two velocity obstacle, or (b) the projection of the preferred velocity onto the closest leg of each velocity obstacle. All points that are within another obstacle are discarded, and from the remaining set the one closest to the preferred velocity is selected. Figure 2.5a shows the graphical interpretation of the algorithm.

With ORCA, the VOs are translated into half-planes which constrain the velocity space into a convex space. The optimal velocity is then in this space and linear programming is used to find the optimal solution for the current situation. An example is

shown in Figure 2.5b.

Another method is to generate possible sample velocities based on the motion model of the robots and test whether these velocities are collision free and how well they are suited. Each sample gets a score according to one or multiple cost functions, i.e. distance to current and preferred velocities and whether it is inside a velocity obstacle or not as shown in Figure 2.5c. The velocity samples should be limited to the velocities that are achievable in the next time-step. If the velocity is not holonomic, the samples can be translated to approximate holonomic velocities as presented in Section 2.1.2. We rollout a trajectory using the current velocity sample and then use the position to calculate the approximate holonomic velocity and the corresponding tracking error.

2.2 Decision Making and Task Allocation

Another importance aspect for Single- and Multi-Robot Systems concerns the decision making on a different level. Namely, how does the robot decide which action to take? This could mean for instance which task to fulfil, or in which direction to drive. This problem of *multi-robot coordination* can be tackled in various ways.

We will look into the Markov Decision Processes (MDPs) (Bellman, 1957a) framework for single agents and its extensions to multiple agents. In these decision theoretical frameworks, usually the term *agent* is used to denote an acting and decision making entity. In our case, when we talk about agents, we generally refer to a robot. However, the agent could also be a *meta-agent* that controls multiple robots at once, or on a lower scale, each actuator of the robot, the wheels could be independently modelled as an agent. This framework models the world as a Markov Process and the actions of the agents are transitions in the process. A major advantage of this framework is that optimality is a well-defined concept and there are many algorithms that are proven to yield optimal policies. Unfortunately, in many cases, if the problem is modelled as an MDP, the state and action space are so large that optimally solving such a problem is computationally unfeasible.

An existing framework for task allocation for robots is the so-called Multi-Robot Task Allocation (MRTA) taxonomy (Gerkey and Matarić, 2004). This taxonomy is a general framework which covers multiple different instances of the MRTA problem. It views the problem of *multi-robot coordination* more as an *Optimal Assignment Problem (OAP)*. Thus, while related, the idea behind this framework is fundamentally different from the MDP-based framework. In the following, we will describe the two frameworks and present common solution methods.

2.2.1 Markov Decision Processes

In their original form, MDPs model single agent decision-making, where the current state is Markovian, i.e. transition probabilities only depend on the current state. Thus, the current state captures all information necessary to select the best possible action.

Formally, a MDP is defined as follows:

Definition 2.1. A *MDP* is defined as a tuple $\langle \mathcal{S}, \mathcal{A}, T, R \rangle$, where:
$\mathcal{S}$ is the finite set of states s of the environment;
$\mathcal{A}$ is the action space, i.e. the set of actions;
T: $\mathcal{S} \times \mathcal{A} \times \mathcal{S}$ is the transition probability function specifying $T(s, a, s') = P(s'|s, a)$;
R: $\mathcal{S} \times \mathcal{A} \to \mathbb{R}$ represents the reward function. It specifies the cost or utility that an agent receives when performing a certain action a in a given state s.

The reward function is usually represented as $R(s, a)$, while sometimes the resulting state is taken into account as $R(s, a, s')$. Nevertheless, we can enforce the $R(s, a)$ notation by calculating

$$R(s, a) = \sum_{s' \in \mathcal{S}} T(s, a, s') R(s, a, s'). \tag{2.5}$$

The core objective in an MDP is to find a policy π for the decision making agent, such that given any state s, the function $\pi(s)$ returns an action a. The goal is to find the optimal policy π^*, such that it maximises the expected future rewards. This expected future rewards can be the average reward over time or the total reward, which is the sum of all future rewards. When using the total rewards, in many cases, the future rewards are discounted by a factor γ in order to prevent the robots to *overvalue* future rewards, as they can also be much more uncertain. In our case we will focus on the (discounted) total rewards.

The policy can be over a limited number of steps in the future, which is commonly referred to as *horizon* h, or unlimited steps, or until a terminal state has been reached, which is commonly used in games, when the agent has won or lost the game.

The policy can either be a deterministic mapping from a state to an action $\mathcal{S} \to \mathcal{A}$, or a stochastic policy, mapping a state to a probability distribution over the possible actions. It has been shown that in a MDP with discounted rewards, there exists at least one optimal deterministic policy (Bellman, 1957a).

In the following section, we describe some common solution methods for finding a (close to) optimal policy π in more detail.

2.2.2 Solution Methods for Markov Decision Processes

As described before, the objective of any agent in an MDP is to maximise its rewards or to minimise its costs depending on the nature of the problem.

Since the transitions in an MDP can be stochastic, we try to optimise the expected sum of (discounted) future rewards. Formally, this can be defined as follows:

$$E_\pi = \sum_{t=0}^{h-1} \gamma^t R(s_t, \pi(s_t)), \tag{2.6}$$

Where E_π defines the expected value when the agent is following the policy π, h is the horizon (which could be infinite) and $\gamma \in [0, 1]$ is the discount value. In finite horizon problems, the discount value is usually set to 1, i.e. using the total sum of future rewards, while for in infinite-horizon problems, the discount value is in [0, 1), to ensure that the series converges.

We can define the *value function* $V : \mathcal{S} \to \mathbb{R}$ of the problem such that it defines a mapping of the current state to the expected value. Where the expected value is the current reward plus the value of any possible successor state given the policy and the transition function of the MDP. This allows us to recursively define the value of a state given a certain policy V^π as:

$$V^\pi = R(s, \pi(s)) + \gamma \sum_{s' \in \int} T(s, \pi(s), s') * V^\pi(s'). \tag{2.7}$$

For the finite horizon case, we can compute the value for a given step $t < h - 1$, as:

$$V_t^\pi = R(s, \pi(s)) + \gamma \sum_{s' \in \int} T(s, \pi(s), s') * V_{t+1}^\pi(s'), \tag{2.8}$$

where $V_t^\pi = 0 \ \forall \ t \geq h$, thus at the last step before the horizon, the value becomes $V_{h-1}^\pi = R(s, \pi(s))$. The easiest way to think about the horizon is that the agent is only interested in the future up to h steps ahead. Afterwards, it does not care anymore.

Dynamic Programming & Value Iteration

If we now want to find an optimal policy π^*, we can use dynamic programming (Bellman, 1957b), to find the optimal value function V_t^*, which is defined as:

$$V^* = \max_{a \in \mathcal{A}} \left\{ (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') * V^*(s') \right\}. \tag{2.9}$$

When the optimal value function is known, the optimal policy π^* can easily be extracted by choosing the action that leads to the highest expected value:

$$\pi^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') * V^*(s') \right\}. \tag{2.10}$$

Similarly, for the finite horizon case, Equation 2.9 and Equation 2.10 can be adapted in a straight forward manner for each step t:

$$V_t^* = \max_{a \in \mathcal{A}} \left\{ (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') * V_{t+1}^*(s') \right\}, \tag{2.11}$$

$$\pi_t^*(s) = \operatorname*{argmax}_{a \in \mathcal{A}} \left\{ (R(s, a) + \gamma \sum_{s' \in \mathcal{S}} T(s, a, s') * V_{t+1}^*(s') \right\}. \tag{2.12}$$

Algorithm 1: Value Iteration

initialise $V(s)$ arbitrarily
repeat
  foreach $s \in \mathcal{S}$ **do**
    foreach $a \in \mathcal{A}$ **do**
      $Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') * V(s')$
    $V(s) = \max_{a \in \mathcal{A}} Q(s,a)$
    $\pi(s) = \text{argmax}_{a \in \mathcal{A}} Q(s,a)$
until *until policy good enough*

The optimality criterion in this case is defined that for every step t, there does not exist another action sequence of length $h - t$ that generates a higher expected reward than $V_t^*(s)$.

An often applied algorithm to compute the optimal value function is called Value Iteration. When applying Equation 2.11 repeatedly over all states, i.e., initialising all states with an arbitrary value and then computing the new value according to Equation 2.11, it has been shown that for $t \to \infty$, it is guaranteed to converge to the optimal value function for the problem according to an optimal stationary policy (Puterman, 1994). Algorithm 1 summarises the approach. In this case a so-called *Q-table* is used which stores the values of a specific state-action pair given a value function V, $Q : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, which is called a *Q-value*:

$$Q(s,a) = R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') * V(s'). \tag{2.13}$$

For finite horizon problems, the number of iterations needed is equal to the horizon h. In looser terms, each iteration takes the value of another step into account. Since $V_t^\pi = 0 \; \forall \; t \geq h$, after h steps the additional value is 0, and does not affect the value anymore. Unfortunately, for the optimal policy, all intermediate steps have to be saved, as shown in Equation 2.12, the policy depends on the current step t and the value of V_{t+1}^*. In infinite horizon problems, only the final optimal value function has to be stored, as can be seen by Equation 2.10.

If the model of the MDP is known beforehand, the optimal value function can be calculated *offline*. This means during execution, the agent only needs to lookup the corresponding action according to the optimal policy π^*. The agent will never deviate from its policy, and always choose the same action if the same state is encountered. This is commonly referred to *offline planning* in an MDP.

Another offline planning method is called *Policy Iteration*. Instead of finding the optimal value function, it operates directly on the policy. We start with an arbitrary policy π. We can calculate the value function of this policy by solving a set of linear

equations such that Equation 2.7 holds. Afterwards, the policy is improved by applying:

$$\pi'(s) = \underset{a \in \mathcal{A}}{\operatorname{argmax}} \left\{ (R(s,a) + \gamma \sum_{s' \in \mathcal{S}} T(s,a,s') * V^{\pi}(s') \right\}. \tag{2.14}$$

This is repeated until the policy does not change anymore. It has been shown that this methods is guaranteed to converge to an optimal policy (Puterman, 1994) in finite number of steps, since there are at most $|\mathcal{A}|^{|\mathcal{S}|}$ discrete policies and at each step the policy is guaranteed to strictly improve.

Learning in Markov Decision Processes

Another option to make decisions in a MDP is through *learning* (Kaelbling et al., 1996; Sutton and Barto, 1998). This means that the agent takes action and interacts with the environment in run-time (i.e. *online*) or in a simulation of the environment for fast iterations. The agent observes the rewards and adapts its actions to optimise the future rewards. Therefore, this type of learning is also referred to as *Reinforcement Learning*, since the agent learns from its reinforcements from the environment.

One of the most common forms of learning is using temporal difference learning, also refereed to as *TD-learning*. The idea is to bootstrap the actual value function by using estimations of itself. TD(λ) was introduced by (Sutton and Barto, 1998). The parameter $\lambda \in [0,1]$ describes a so-called eligibility trace parameter, i.e. how much the more distant estimations have influence on the current estimation. This influence is can be expressed for each state s at time t as follows:

$$e_t(s) = \begin{cases} \gamma\lambda e_{t-1}(s) & \text{if } \ s \neq s_t \\ \gamma\lambda e_{t-1}(s) + 1 & \text{otherwise} \end{cases} \tag{2.15}$$

The γ parameter is a discount rate. For $0 < \lambda < 1$, the influence decreases over time. In TD(1), the current estimation will also be used to update all preceding estimations. This is also called Monte Carlo Reinforcement Learning. We will focus on the so called TD(0) update rule. In this case, the value function update is the following:

$$V(s) = (1-\alpha)V(s) + \alpha \left(R(s, \pi(s)) + \gamma * V^{\pi}(s') \right), \tag{2.16}$$

where α is the learning rate, which controls the impact of a new observation. It acts as an exponential smoothing over the observed values.

The idea is similar to Value Iteration (cf. Algorithm 1) and Policy Iteration, however the difference is that the value is drawn from the experience in the environment, rather than computing from a known model.

Additionally, we can use the *Q-values*, as introduced in Equation 2.13, and further define $Q^*(s,a)$ as the expected discounted reward when choosing actions optimally. Thus, since $V^*(s)$ defines the optimal value of a state, assuming the best possible actions,

Algorithm 2: Q-learning

initialise $Q(s,a)$ and $\pi(s)$ arbitrarily
Set initial state s
repeat
  Select action a based on current policy π and action selection policy
  Take action a and observe reward $R(s,a)$ and new state s'
  $Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma * \max_{a' \in \mathcal{A}} Q(s',a')\right)$
  foreach $s \in \mathcal{S}$ **do**
    $\pi(s) = \text{argmax}_{a \in \mathcal{A}} Q(s,a)$
  $s = s'$
until *until policy good enough*

it follows that $V^*(s) = \max_{a \in \mathcal{A}} Q^*(s,a)$ and the optimal policy can be defined as $\pi^*(s) = \text{argmax}_{a \in \mathcal{A}} Q^*(s,a)$. Following the same notations, we can write an update rule for the Q-values similar to Equation 2.16:

$$Q(s,a) = (1-\alpha)Q(s,a) + \alpha\left(R(s,a) + \gamma * \max_{a' \in \mathcal{A}} Q(s',a')\right), \tag{2.17}$$

which is commonly referred to as *Q-learning* (Watkins, 1989).

It has been shown to converge to the optimal policy, as long as all state-action pairs are visited often enough. Thus, the agent has to explore the environment to ensure that the Q-values converge, however the agent also needs to exploit the knowledge that it already has (in form of the current Q-values). Hence, commonly in learning approaches a stochastic policy is applied that balances random actions with the currently best evaluated action.

A common action selection policy is ε-greedy, which chooses the currently best evaluated action $\text{argmax}_{a \in \mathcal{A}} Q(s,a)$ with a probability of $1-\varepsilon$ and a random action otherwise. Another possibility is to choose the action according to the Boltzman exploration algorithm:

$$P_s(a) = \frac{e^{\frac{Q(s,a)}{T}}}{\sum_{b \in \mathcal{A}} e^{\frac{Q(s,b)}{T}}}, \tag{2.18}$$

where $P_s(a)$ defines the probability of choosing action a in a state s and T is the so-called *Boltzman Temperature*, which typically decreases while the learning progresses. When $T = 0$, the action selection is equivalent to a pure greedy action selection, and with $T \to \infty$, the action selection is purely random. For $T \in (0, \infty)$, actions with a higher value have a greater chance to be selected than lower valued actions.

Algorithm 2 summarises the Q-learning approach. As can be seen, the algorithm does not depend on any knowledge of the model, therefore it is also referred to as *model-free*. Also the Q-learning update rule operates only on the Q-values and assumes the maximum value for the future states. This is called *off-policy*. Some *on-policy* algorithms

are $TD(\lambda)$, SARSA and actor-critic methods. For more information, we refer the reader to (Kaelbling et al., 1996; Sutton, 1984; Sutton and Barto, 1998).

2.2.3 Extensions for Planning with Multiple Agents

When planning in a multi-agent system, the MDP framework can be extended to a Multi-Agent Markov Decision Process (MMDP) in a straight forward manner (Boutilier, 1996). In this case a team of agents has to maximise a joint global reward function. It is formalised as follows:

Definition 2.2. A *MMDP* is defined as a tuple $\langle \mathcal{D}, \mathcal{S}, \mathcal{A}, T, R \rangle$, where:
$\mathcal{D} = \{1, \ldots, n\}$ is the set of n agents;
$\mathcal{S}$ a finite set of states s of the environment;
$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ the set of joint actions $a = \langle a_1, \ldots, a_n \rangle$;
T the transition probability function specifying $P(s'|s, a)$;
$R(s, a)$ the immediate reward function.

Similar as in a MDP, a (joint) policy $\pi : \mathcal{S} \to \mathcal{A}$ in an MMDP maps states s to joint actions a, and is equivalent to a tuple of individual policies $\pi_i : \mathcal{S} \to \mathcal{A}_i$. The Q-value of (s, a) under policy π is defined as the expected sum of rewards when executing a in s and following π afterwards.

Solving MMDPs can be done in a similar fashion as (single-agent) MDP as presented in the previous sections. For instance, a MMDP can be seen as a single agent MDP, in which we define one *meta-agent* that plans for all other agents, which select their action accordingly during execution. Otherwise, since the global state is known to all agents, each agent can plan on its own, but still has to account for all other agents during its planning.

An MMDP is called *factored* if its state space is spanned by a set of state variables. This means, the state space can be decomposed into several independent variables, for instance multiple agents move within an environment and their transitions are independent from the other agents. More specifically, the state space is defined by a number of components, i.e. $\mathcal{S} = \mathcal{S}_0 \times \mathcal{S}_1 \times \cdots \times \mathcal{S}_n$. This feature can be exploited in the planning process, especially if the transitions and rewards can also be factored in a similar fashion. Nevertheless, since the number of joint actions is exponential in the number of agents and the number of states is exponential in the number of factors (itself usually dependent on the number of agents), an exact solution in this framework is usually, intractable for even small problems in practice.

The most general extension of the MMDP framework are the Decentralized Partially Observable Markov Decision Processes (Dec-POMDPs). In this setting, the agents cannot directly observe the state, but they only perform observations in order to reason about the state. The agent only has a partial perception of the global state, but still have to maximise the global reward. Formally, a Dec-POMDP is defined as a follows:

Definition 2.3. A *Dec-POMDP* is defined as a tuple $\langle \mathcal{D}, \mathcal{S}, \mathcal{A}, \mathcal{Z}, T, O, R \rangle$, where:
$\mathcal{D} = \{1, \ldots, n\}$ is the set of n agents;
$\mathcal{S}$ a finite set of states s of the environment;
$\mathcal{A} = \mathcal{A}_1 \times \cdots \times \mathcal{A}_n$ the set of joint actions $a = \langle a_1, \ldots, a_n \rangle$;
$\mathcal{Z} = \mathcal{Z}_1 \times \cdots \times \mathcal{Z}_n$ the set of joint observations $z = \langle z_1, \ldots, z_n \rangle$;
T the transition probability function specifying $P(s'|s, a)$;
O is the observation probability function specifying $O(z|s, a)$;
$R(s, a)$ the immediate reward function.

In this case, the policy for an agent i is determined also by its history of previously chosen actions and observations $h_i = a_i(0), z_i(1), \ldots, a_i(t-1), z_i(t$. Thus the policy in the finite horizon case for agent i, is a mapping $\pi_i : \mathcal{H}_i \to [0, 1]$ such that for all histories $h_i \in \mathcal{H}_i$, it holds $\sum_{a_i \in \mathcal{A}_i} \pi_i(h_i, a_i) = 1$. In this setting, the agents keep track of a belief vector, which denotes agents' probabilities of being in a certain state.

Even though this model is the most general form, in terms of representation possibilities, exact solution can only be computed in very few instances, and in most cases only heuristic and approximate models can be applied (Seuken and Zilberstein, 2008).

In a special case, if the agents have full local observability, i.e. the agents are able to observe their own local state deterministically, the Dec-POMDP reduces to a Decentralized Markov Decision Process (Dec-MDP). This means that if the agents are able to share their local state among each other, they can deduce the exact state of the system. If each agent can individually observe the full global state individually, this problem reduces to an MMDP.

To summarise, optimally solving any large MMDP or Dec-POMDP is a computationally costly problem in practice and not suitable for online planning for a team of agents. Therefore, in this thesis, we will focus on approximations and heuristics that exploit the underlying structure of the problem in order to achieve feasibility for solving these kinds of problems in an online fashion.

2.2.4 Monte Carlo Tree Search

One possible heuristic method that is successfully used within the MDP domain is MCTS. It is a simulation based search algorithm (Kocsis and Szepesvári, 2006; Coulom, 2007), which builds up a search tree based on simulations. It has initially been applied in games, i.e. it has been very successful in general game playing (Browne et al., 2012) and computer Go—it is the basis for AlphaGo, the first AI program to beat a professional Go player on the full size board and without handicap (Silver et al., 2016). But in recent years, MCTS has also been applied to many non-game domains such as large MDPs (Silver and Veness, 2010).

The main intuition behind MCTS is that by using Monte Carlo simulations to quickly sample thousands of possible trajectories, we can achieve good approximations of the

Algorithm 3: UCT based MCTS

Input : $\mathcal{S}$: Current state
n: number of simulations
d_{max}: maximum depth
w: sampling width
γ: discount

Output: a: action for agent

Algorithm `MCTS_Search()`
- initialise $Q(s, a)$ arbitrarily
- **for** $i = 1$ *to* n **do**
  - $s_0 \leftarrow \mathcal{S}$
  - $d \leftarrow 0$
  - `SimulateV(`s_0`)`
- $a \leftarrow \max_a Q(s, a)$
- **return** a

function `SimulateV(`s`)`
- **if** $d = d_{max}$ **then**
  - **return** *0*
- $a \leftarrow \text{argmax}_a \left(Q(s, a) + C\sqrt{\frac{\texttt{ln(N(}s\texttt{))}}{\texttt{N(}s,a\texttt{)}}} \right)$
- $r \leftarrow$ `SimulateQ(`s, a`)`
- **return** r

function `SimulateQ(`s, a`)`
- $s', r \leftarrow$ `SimulationStep(`$s,\ a$`)`
- $d \leftarrow d + 1$
- **if** `Visited(`s'`)` **then**
  - $r' \leftarrow$ `SimulateV(`*s'*`)`
- **else**
  - $r' \leftarrow$ `Rollout(`*s'*`)`
- $d \leftarrow d - 1$
- $r \leftarrow r + \gamma * r'$
- $Q(s, a) \leftarrow$ `UpdateReward(`$Q(s, a), r$`)`
- **return** r

values of possible actions. While doing these Monte Carlo simulations, a search tree, which stores statistics used to guide the search, is built incrementally starting from just a root node by exploring the most promising actions first. To achieve this, the search tree keeps track of how often each node has been visited, and which estimated value can be achieved when choosing each action at each node. When 'inside' the search tree, the statistics are used to select the most promising actions, when 'outside' the tree, action selection is guided by (computationally cheap) rollout policies.

MCTS starts every Monte Carlo simulation at the root node, corresponding to the current state. Actions are chosen according to a tree policy until the resulting state has not been visited before. The search leaves the tree, and a rollout policy is applied until a stopping condition is met, which can for instance be the end of the simulation or a given

search depth. The value resulting from this rollout is back-propagated through the tree, and the values for each parent node are updated. Thus, the value approximations of all visited states and actions are improved. This process is repeated until the specified planning time or number of simulations are reached. A commonly used tree policy is UCB1 (Kocsis and Szepesvári, 2006), resulting in the UCT algorithm. UCT has been shown to converge to the optimal policy in single-agent and two-agent zero-sum problems (Kocsis et al., 2006).

In order to achieve a high number of simulations, MCTS requires a simulator that can quickly generate a successor state and a reward for a given action. This simulator can for instance be based on an MDP, since the transition function in an MDP specifies the probabilities for any given successor state give the current state and action.

Algorithm 3 shows the implementation that can be used when using MCTS in an MDP environment. The search is started with the initial state, and the simulations are performed. The `SimulateV` and `SimulateQ` functions are used to approximate the Q-values and V-values, similar as in Q-learning (cf. Algorithm 2). In the `SimulateV` function a new action is chosen according to the equation:

$$a \leftarrow \underset{a}{\operatorname{argmax}} \left(Q(s,a) + C\sqrt{\frac{\texttt{ln(N(}s\texttt{))}}{\texttt{N(}s,a\texttt{)}}} \right),$$

in which the first part is the same as in Q-learning, while the second part is the UCT formula, which has the parameter C as exploration constant, which controls the likelihood that a less explored action is chosen, even if it has a lower value. In practice, this parameter is chosen empirically depending on the domain. In the `SimulateQ` function, the effect of the action is simulated, i.e. in the MDP environment, the state and action result in an observed immediate reward r and a new state s' and then the delayed reward of the chosen action is evaluated. Either we have seen the new state s' before, in which case we can progress deeper into the tree, or we reach the end of the currently built tree and we perform a rollout based on the current state. The rollout yields the delayed reward r', which we add to the total reward with a discount factor γ. The `UpdateReward` function then updates the corresponding value in the Q-table.

Thus in summary, MCTS can be seen as a form of Q-learning which only starts the search from the current state and simulates thousands of different possibilities, while keeping track of the resulting values. Afterwards, the action which yields the best result is chosen.

2.2.5 Multi-Robot Task Allocation and Market-based Approaches

While the MDP framework has been extensively developed in the Multi-Agent System (MAS) community, these methods can be applied to MRS. As mentioned before, an agent is an decision making entity, so we can see a robot as an agent and directly apply the presented methods, as long as the problem can be modelled within the constrains of the framework.

The MDP-framework and all its variants are developed from the perspective of actions, i.e. which action should the agents take given the current state. These actions are usually very fine-grain, for instance moving a single step in a grid-world, performing a task, or making an observations, leading to a computational high complexity.

A different perspective is the Multi-Robot Task Allocation (MRTA) framework (Gerkey, 2003). It introduces a formal taxonomy in multiple dimensions with which the MRTA problem can be defined (Gerkey and Matarić, 2004). For instance, the tasks can be executable only by *single-robot (SR)* or must have *multiple robots (MR)* to be completed. The robots can be *single-task robots (ST)* or *multi-task robots (MT)*, which means that robots can either work on only one task at a time or on multiple tasks at the same time.

Also the tasks can be *static (SA)*, which means that the tasks are either all known beforehand or they appear during the execution, i.e. *dynamic (DA)*. This taxonomy has been extended to include temporal and ordering constraints, such that a task may be dependent on others to be completed beforehand, for instance, cleaning some debris before being able to enter, i.e. *constraint (CT)* or the tasks are *independent (IT)* (E. Schneider et al., 2015; Nunes et al., 2017).

In this framework, it is shown that the problem of MRTA can be cast in various different forms. For instance, the problem can be seen as an Optimal Assignment Problem (OAP) (Gale, 1960), when considering the class of SR-ST-SA-IT, i.e. robots can do one task at a time, the tasks only need one robot to be performed, the tasks are known beforehand and they can be independently performed. These problems have been well studied in game theory and operations research in the context of personnel assignment.

Formally, the MRTA can be defined as an OAP follows:

Definition 2.4. A MRTA is defined as a tuple $\langle \mathcal{D}, \mathcal{T}, \mathcal{U} \rangle$, where:
$\mathcal{D} = \{d_1, \ldots, d_n\}$ is the set of n robots;
$\mathcal{T} = \{(t_1, w_1), \ldots, (t_i, w_i)\}$ a finite set of i tasks with weight w;
$\mathcal{U} = \{U_{1,1}, \ldots U_{n,i}\}$ is the set of utilities that defines the value for each robots n and task i.

This assumes that each robot n in capable of executing at most one task at any given time and each task (t_i) requires exactly one robot to execute it. While these assumptions are somewhat restrictive, they are necessary to classify the problem as an OAP. Also while they can be relaxed, in many cases these assumptions do apply.

The main objective of this problem is to find the optimal allocation of the tasks to the robots, where an allocation is defined as the set of robot-task pairs (d_n, t_i). In the OAP definition, there are at most $1 \leq k \leq \min(n, i)$ robot task pairs, i.e. each robot has at most one task assigned. The value of this assignment is now defined as the weighted sum of the tasks priorities and the robots values:

$$U = \sum_{l=1}^{k} U_{n_l, i_l} w_{i_l}. \tag{2.19}$$

These kind of problems can be solved using for instance integer linear programming (Gale, 1960). While integer linear programs are NP-hard, many problems like the OAP exhibit a special structure that can be exploited to facilitate their solution (Ahuja et al., 1988). The MRTA problem can be cast as an integer linear program as follows (Gerkey, 2003): Find m^2 integers $a_{n,i}$ that are either 0 or 1, that maximise:

$$\sum_{n=1}^{m}\sum_{i=1}^{m} a_{n,i}U_{n,i}w_i, \tag{2.20}$$

subject to:

$$\sum_{n=1}^{m} a_{n,i} = 1, 1 \leq i \leq m, \tag{2.21}$$

$$\sum_{i=1}^{m} a_{n,i} = 1, 1 \leq n \leq m. \tag{2.22}$$

The sum (Equation 2.20) maximises the overall utility, while the constraints (Equation 2.21 and Equation 2.22) enforces the assumptions that at most one task is assigned to one robot.

Another common method for MRTA is using a *marked based approach* (Shapley and Shubik, 1971). A popular form of this solutions are *auctions*. In this case we assume *task-market*. The robots have to bid for tasks and there is a centralised broker or auctioneer which sells the tasks.

A common algorithm is the Sequential Single Item (SSI) algorithm (Koenig et al., 2006). The general idea is that all tasks are offered at the same time to all robots. The robots have to compute bids for all tasks and the auctioneer robot selects the winner as the robot with the lowest bid. The winning robot update its bids according to the new location of the task and the next round is performed.

Algorithm 4 shows the general idea of the algorithm. The function `cost(`n, t`)` defines the cost of a task t for a given robot n. This could be for instance the distance the robot has to travel to the task and the cost of the robot to perform the task.

Some other variants of SSI approach include the Ordered Single Item (OSI)-auction (E. Schneider et al., 2014) in which the tasks are placed in an ordered list, and only one task is offered to the robots at the same time and the Parallel Single Item (PSI)-auction (Koenig et al., 2006), which assigns *all* tasks in one round.

These kind of auction based approaches are often used in MRS- domains (Choi et al., 2009; Nanjanath and Gini, 2010; Amador et al., 2014; E. Schneider et al., 2015). Unfortunately, (E. Schneider et al., 2015) found "that the advantages of the best auction-based methods are much reduced when the robots are physically dispersed throughout the task space and when the tasks themselves are allocated over time".

It is possible to capture MRTA problems formalised as an MMDP. The joint-actions are the possible allocations of the tasks, the rewards are the costs incurred for the

Algorithm 4: Sequential Single-Item Auctions

Input :
$\mathcal{D}$: set of other Agents
$\mathcal{T}$: set of Tasks
Output:
$T(i)$: allocation of tasks for all $i \in \mathcal{D}$
foreach $i \in \mathcal{D}$ **do**
  $T(i) = \emptyset$
repeat
  foreach $i \in \mathcal{D}$ **do**
    foreach $t \in \mathcal{T}$ **do**
      `bid(`i`,` t`) = cost(`i`,` t`)`
  $(i_{win}, t_{win}) = \text{argmin}_{i \in \mathcal{D}, t \in \mathcal{T}}$ `bid(`i`,`t`)`
  $\mathcal{T} = \mathcal{T} \setminus \{t_{win}\}$
  $T(i_{win}) = T(i_{win}) \cup \{t_{win}\}$
until *until* $\mathcal{T} = \emptyset$

robots for performing the tasks and the transition function expresses the changes after each step of the assignment. Afterwards, the MMDP solution methods can be applied. Furthermore, with the MMDP it is possible to capture much more detail, since we do not need to look at the task performing level, but we can also optimise along more *low level* actions, i.e. moving around, so we will focus on using MMDP for this research.

2.3 Robotic Platforms and Core Components

In this research we have used two main robotic platforms, the Turtlebot and the youBots (Bischoff et al., 2011). While these robots are commonly available[1], we have made several modifications to fit the robots to our needs. These are presented in the Appendix A.

All of our robots are running the Robot Operating System (ROS) framework (Quigley et al., 2009). ROS is designed as middle-ware and framework for robotic platforms. Additionally, it is an open source toolkit to prevent "reinventing the wheel". One of the primary goals stated on the ROS website is to "support code reuse in robotics research and development"[2]. Thus, our approach is not limited to the two robot platforms that are presented in the following, but any robot running ROS can be adapted to work in our approach, given that it has similar capabilities. The code for this work can be found on http://github.com/smARTLab-liv/.

A prerequisite for any autonomous robot is to have an idea of the environment and where the robot is located with respect to a given reference frame. For instance, the VO paradigm assumes that the relative positions of the robots and velocities are known, which is straight forward if the robots are running in simulation. Unfortunately, in the real world this is not as simple.

[1] As of 2017, the youBot is not produced anymore
[2] For more information see: http://www.ros.org/.

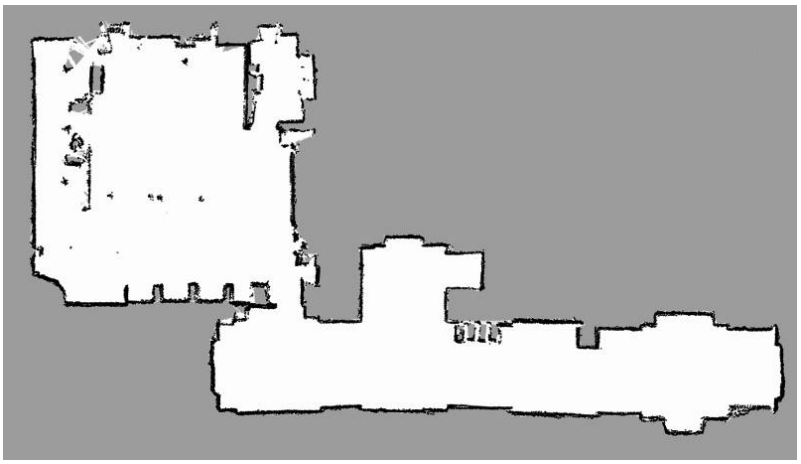


FIGURE 2.6: A map created by gmapping from the smARTLab laboratory and the adjacent corridor.

Thus, the robots have to create a map which provides the reference frame. This is usually referred to as the Simultaneous Localization and Mapping (SLAM) problem. This is a complex problem, since for building a map, the robot needs to have an idea of where it is located according to a reference frame, but for this localisation the robot needs a consistent and accurate map. This dependency between localisation and map building makes the SLAM problem very difficult. One approach for tackling this problem is called gmapping[3]. As this method is only used to obtain a map, a detailed description is out of the scope of this work. For more information about the approach, we refer the reader to (Grisetti et al., 2007). Any other SLAM method which provides a grid map as output can be used as for instance HectorSLAM (Kohlbrecher et al., 2011). An example of a map created by gmapping is shown in Figure 2.6.

After the map is built, it can be used for by the robots to for localisation and navigation. In the following, we will introduce the methods used in our work for localisation and navigation.

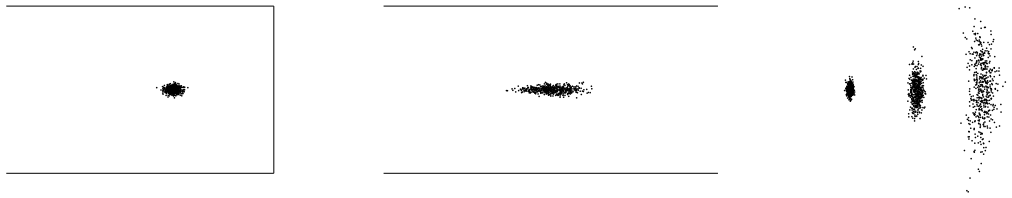
2.3.1 Localisation

The localisation method employed in our work is based on sampling and importance based resampling of particles, in which each particle represents a possible pose and orientation of the robot. More specifically, we use the Adaptive Monte Carlo Localization (AMCL)[4] method, which dynamically adapts the number of particles (Fox, 2003).

AMCL (also known as a particle filter localisation), is a widely applied localisation method in the field of mobile robotics. It can be generalised in an initialisation phase and two iteratively repeated subsequent phases, the prediction and the update phase.

In the initialisation phase, a particle filter generates a number of samples N, which are uniformly distributed over the whole map of possible positions. In the so-called

[3] http://wiki.ros.org/gmapping
[4] http://wiki.ros.org/amcl



(a) End of corridor (b) Long corridor (c) Open space

FIGURE 2.7: Typical particle filter situations. (a) A well localised robot at the end of a corridor resulting in a particle cloud with small variance. (b) In an open ended corridor the sensor only provides valid readings to the sides, resulting in an particle cloud elongated in the direction of the corridor. (c) In an open space no sensor readings result in a particle cloud driven purely by the motion model.

2.5D case, every particle (s^i, w^i) has a x- and y-value and a rotation $s^i = (\hat{x}, \hat{y}, \hat{\theta})$ and a weight (w^i).

The first iterative step is the prediction phase in which the particles of the previous population are moved based on the motion model of the robot, i.e. the odometry. Afterwards, in the update phase, the particles are weighted according to the likelihood of the robot's measurement for each particle. The new weight (w_k^i) is the probability of the actual sensor measurement (z_k) given the particles position (s_k^i) at time k as shown below:

$$w_{k+1}^i = p(z_k | s_k^i). \tag{2.23}$$

Since w is a probability distribution, the weight for each particle is re-normalised after each update:

$$w_k^i = \frac{w_k^i}{\sum_i w_k^i}. \tag{2.24}$$

Given this weighted set of particles the new population is resampled in such a way that the new samples are selected according to the weighted distribution of particles in the old population. For further details we refer to (Fox, 2003).

In this work, AMCL is not used for global localisation, but rather initialised with a location guess that is within the vicinity of the true position given by the human operator. This enables us to use AMCL for an accurate position tracking without having multiple possible clusters in ambiguous cases. If there are cases with multiple clusters, we will use only the most likely cluster and discard the rest of the particles.

Unfortunately, a common problem occurs if the environment looks very similar along the trajectory of the robot, e.g. a long corridor; or a big open space with only very few valid sensor readings. In these cases, particles are mainly updated and resampled according to the motion model leading to the situations shown in Figure 2.7. These

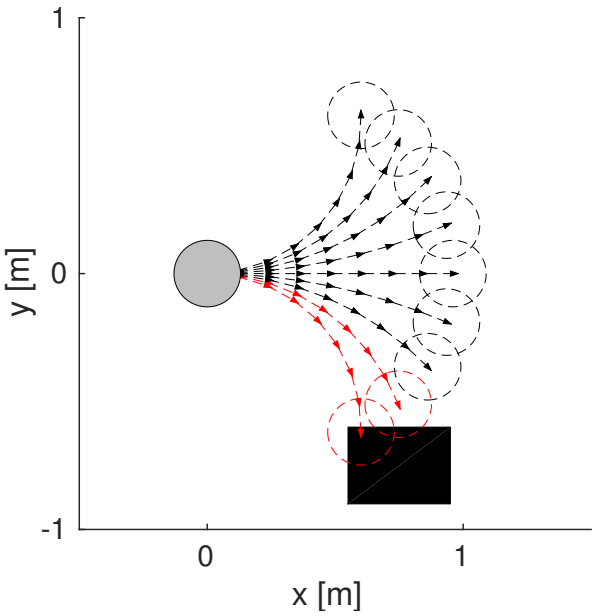


FIGURE 2.8: Dynamic Window Approach (DWA) generates various sample control inputs and uses forward simulations in the configuration space to detect if the given combination of control inputs leads to a collision. In this example, the lower two trajectories lead to a collision and will be excluded.

cases have to be kept in mind when navigating, since simply trying to encompass all particles by enlarging the virtual footprint of the robots might lead to unintended results in which the robots cannot navigate anymore.

2.3.2 Navigation using the Dynamic Window Approach and a Global Plan

When a robot is able to successfully localise itself in an environment, (as, for instance, when using AMCL with a pre-recorded map as explained in the previous section) to be autonomous, the robot has to be able navigate to a given goal location.

A commonly used approach for this navigation is the DWA (Fox et al., 1997) together with a global path planning algorithm. This global path planning algorithm is usually a Dijkstra or A* (Hart et al., 1968) type of search based on the known grid map which is, for instance, created using gmapping or HectorSLAM as described in the previous section. Detected obstacles are marked in this map when they are seen by one of the robot's sensors. In order to create an environment for fast and efficient search, the obstacles that are marked in the map are inflated by the robot's circumscribed radius. This simplifies the problem since the robot can now be seen as a point.

After the global path is found, a local control algorithm such as the previously mentioned DWA has the task to follow this path towards the goal while staying clear of obstacles. DWA creates samples in the control space of the robot. More specifically, it creates samples in every possible velocity dimension that the robot is actuated in. For example, a differential drive robot can be actuated in linear velocities in *x-direction*

(forward and backward) and angular velocities, and, a holonomic robot, can additionally be actuated in linear velocities in *y-direction (left and right side-ways)*. These samples are created based on the current velocity and the dynamic constraints of the robots from which the name *dynamic window* is derived.

When the velocity samples have been created, the robot uses a forward simulation to predict the effect of the given velocity in the configuration space. In other words, the robot simulates the trajectory, if the given velocity would be commanded. Afterwards, this trajectory is scored based on various cost functions. For instance, the robots footprint is imposed on each point in the simulated trajectory and if the robot is in collision at any point, the trajectory is excluded. Other cost functions are, for instance, the distance to the goal location and the distance to the given path. Figure 2.8 shows a graphical interpretation of the approach for a differential drive robot.

2.4 Summary

In this chapter, we have introduced the main theory that will form the basis for the remainder of this work. The VO paradigm with its extensions has been presented as an approach on how to deal with dynamic obstacles. Additionally, the MDP framework and the MRTA taxonomy are explained on which the developed algorithms are based upon. In the final part of the chapter, we introduce the robotic platforms that have been used and the core components that are needed for an autonomously acting robot.

Chapter 3

Collision Avoidance in Shared Workspaces

Current research in mobile robotics focuses more and more on enabling robots and humans to share a common workspace. As introduced in Chapter 1, a well known research initiative in this direction is the *Factory of the Future*, also known as *Industry 4.0*, which has the goal to develop smart factories with networked tools, devices, and mobile manipulation platforms (e.g. the KUKA youBot).

Nowadays, robots in manufacturing are typically not designed to be mobile and human-safe. They are placed inside cages and operation is interrupted as soon as a human enters the safety zones. Current solutions for mobile robots in manufacturing settings are restricted to predefined paths, e.g., tracks on the floor, or restricted to movement in a grid to ensure easy navigation. Humans are not allowed to enter the navigation zone of the robots in order to ensure safety.

Relying on predefined paths and grids for navigation is too restrictive and does not allow for a flexible and generally applicable setup of a mobile Multi-Robot System (MRS). Ideally, robots should be able to plan their paths through any open space and ensure safety without any external limitations such as restricted zones. Additionally, in an unstructured work-space there are no traffic rules that direct the navigation. To safely navigate in such a shared multi-robot and human setting, the robot system has to take into account that the surrounding moving 'obstacles' are essentially pro-active agents and thus might aim to avoid collisions.

Although robot localisation is a requirement for multi-robot collision avoidance, most approaches assume perfect sensing and positioning and avoid local methods by using global positioning via an overhead tracking camera - or are purely simulation based. Nevertheless, to be able to correctly perform local collision avoidance in a realistic environment, a robot needs a reliable position estimation of itself and the other agents and humans without the help of external tools. Additionally, MRS in a real-world environment need methods to deal with the uncertainty in their own positions, and the positions and possible actions of the other agents.

In this chapter, we make the following three contributions to this field: First, we show a reliable estimation of the localisation uncertainty using Adaptive Monte Carlo Localization (AMCL), then we combine this with a sampling-based approach to incorporate human avoidance and lastly, by incorporating the commonly used Dynamic Window Approach (DWA) with a global path planner, allows us to handle complex environments with multiple dynamic, i.e. humans and robots, and static obstacles.

In more detail, we show how the distribution of the particle cloud when using AMCL can be used as an estimator for the localisation uncertainty. The approach, which we called Convex Outline Collision Avoidance under Localization Uncertainty (COCALU), uses the localisation uncertainty as an estimator to enlarge the robots' footprints to ensure safe navigation within the vicinity of other robots. The robots share footprint and position information using limited local communication. This assumes that the robots share the same reference frame, and that the robots can communicate with each other in a limited range. These assumptions can be accommodated in many settings, i.e. (local) communication can be realised via radio or WiFi, and the common reference frame is realised by using the same map for all robots.

We introduce a sampling based approach that incorporates human avoidance. By using the sampling based approach together with a more complex evaluation function, more control over the behaviour of the robots is gained. For instance, it is straight forward to discourage robots to pass closely by humans by assigning a high cost, while closely passing by other robots has lower costs.

Lastly, we introduce the combination of the sampling based approach with the DWA (Fox et al., 1997) and a global path planning algorithm. The DWA is commonly used as control algorithm for local control as introduced in Section 2.3.2. It is the standard method which is used on many platforms when using Robot Operating System (ROS) (Quigley et al., 2009). It uses forward simulations of a set of velocity commands, known as trajectory rollouts. In our experiments, we show how the sampling based method can successfully be combined with the DWA approach to ensure good navigation within the proximity of other robots, static obstacles and humans.

The remainder of this chapter is structured as follows. Section 3.1 summarises the related work; Section 3.2 introduces our previous work of combining onboard localisation with the velocity obstacle paradigm. In Section 3.3 we extend the previously introduced algorithm with human detection and a sampling based approach and combine it with the DWA algorithm. Section 3.4 presents the empirical results of the approaches. Section 3.5 concludes the chapter with a summary.

3.1 Related Work

Typically, path-planning methods for navigation are divided into global planning and local control. The global planner searches through the configuration space to find a path from the current location towards the goal location. The task of the local controller is

then to steer free of collisions with any static or dynamic obstacle, while following the global plan to navigate towards a goal location.

3.1.1 Local control

Many approaches for local control make use of the "frozen world" assumption, i.e. that the world is static in each time-step. In (Thrun et al., 2005) a number of probabilistic approaches are presented for a single robot environment. Potential fields are an approach that creates a virtual force-field in the map. Around obstacles it is pushing the robot away, and near the goal, it is pulling the robot towards it. In (Koren and Borenstein, 1991) the limitations of this approach are presented and described. Another approach is the dynamic window approach as described in (Fox et al., 1997). However, all of these approaches lack the possibility to navigate safely within a dynamic multi-robot environment.

In multi-robot collision avoidance research, there is often a centralised controller. For instance, in (Bruce and Veloso, 2006) an approach for safe multi-robot navigation within dynamics constraints is presented. However, these approaches are not robust, since if the centralised controller fails, the whole system breaks. Another common approach is motion planning, which can take dynamic obstacles into account. The main assumption here is that the whole trajectory of the dynamic obstacles is known as in (Ferrara and Rubagotti, 2009).

Another way to ensure collision-free motions is to use formations. For instance, in (Saska et al., 2013) an approach is presented which allows multiple car-like unmanned mobile vehicles to navigate in a common workspace. Formations are very powerful for use-cases in which the robots have to stay together and move towards a common goal. To achieve this goal, the robots are using virtual leaders which the other robots are following. This is in contrast to our approach, in which the robots are able to move around freely in the environment.

In (Althoff et al., 2012) a probabilistic threat assessment method for reasoning about the safety of robot trajectories is presented. Monte Carlo sampling is used to estimate collision probabilities. In this approach, the trajectories of other dynamic obstacles are sampled. This way, a global collision probability can be calculated. This work is closely related to the research done in this paper; however, that approach is probabilistic instead of the geometric representation used for the algorithms we propose.

Recently, in (Bareiss and Berg, 2015), a generalised reciprocal collision avoidance method was introduced. This method uses control obstacles, i.e. it looks which input controls may lead to a collision in the future. This enables planning for any kind of robot where the motion model is known. However, these control obstacles are non-linear making the calculations more complex. Additionally, the work does not consider static obstacles and the experiments rely on an external positioning system.

3.1.2 Collision avoidance in shared workspaces

This work introduces a local collision avoidance approach that deals with the problems of multiple robots sharing the same workspace with or without humans. An overview of existing (global and local) approaches for human aware navigation (Kruse et al., 2013) shows that the main focus of current research is on the comfort, naturalness and sociability of robots in human environments. This usually entails only one robot acting in a group of humans, i.e. as a personal assistant. Our approach however, is aimed at a different distribution of agents, namely many robots navigating together with many humans in the same shared workspace.

An example of the single robot, multi-human navigation approach is the stochastic CAO approach (Rios-Martinez et al., 2012), which models the discomfort of humans and uses the prediction of human movement to navigate safely around people. Another similar approach is described in (Lu, 2014). It is based on layered costmaps in the configuration space and it also describes a user study where gaze-detection was used to determine the intended heading of the humans to update the costs. This layered costmaps idea is similar to the multiple evaluation functions in our approach. However, it is purely based on the configuration space, i.e. it assumes all obstacles to be static. Hence, this approach also does not cover the dynamic nature of moving obstacles as opposed to our presented approach, which uses the velocity space to explicitly model dynamic obstacles.

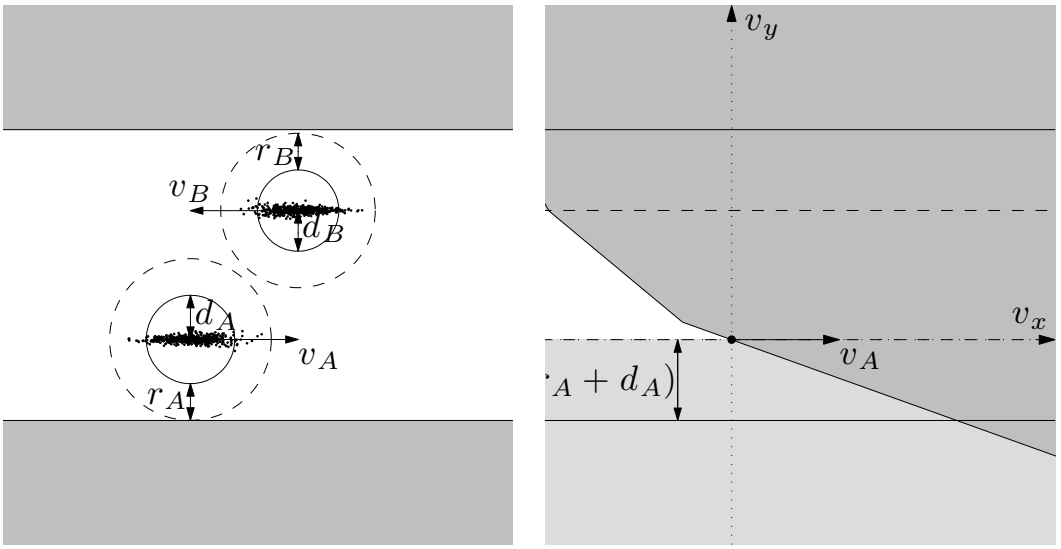
Similarly, the work in (Linder et al., 2016) has the focus on a single robot acting in a multi-human environment. The focus is on tracking and predicting humans and classifying multiple humans into groups. This research is complementary to the work in this paper as it allows the robots to detect and track humans, which is necessary for collision avoidance.

In (Alonso-Mora et al., 2015a), a collision avoidance algorithm for multiple unmanned aerial vehicles (UAVs) is introduced. In that research, a centralised and decentralised convex optimisation approach are explained and the system is integrated with two UAVs flying in close proximity of a human. However, they rely on external positioning in order localise the UAVs and the processing is performed off-board on an external machine.

Other approaches for multi-robot collision avoidance use auctions (Calliess et al., 2012) at a rather high communication overhead, or stigmergy (Theraulaz and Bonabeau, 1999; Lemmens and Tuyls, 2012; Osten et al., 2014), which relies on pheromones that are hard to apply in a real world setting. Additionally, these approaches do not implement robot-human avoidance.

3.2 Convex Outline Collision Avoidance under Localization Uncertainty

In earlier work, *Collision Avoidance under Localization Uncertainty (CALU)* (Hennes et al., 2012) we successfully combined the velocity obstacle approach with onboard



(a) Configuration space (b) Velocity space

FIGURE 3.1: The corridor problem: Approximating the localisation uncertainty (and the footprint) with circumscribed circles, vastly overestimates the true sizes, such that the robots do not fit next to each other. Thus, the Hybrid Reciprocal Velocity Obstacle (HRVO) together with the VO of the walls invalidates all forward movements.

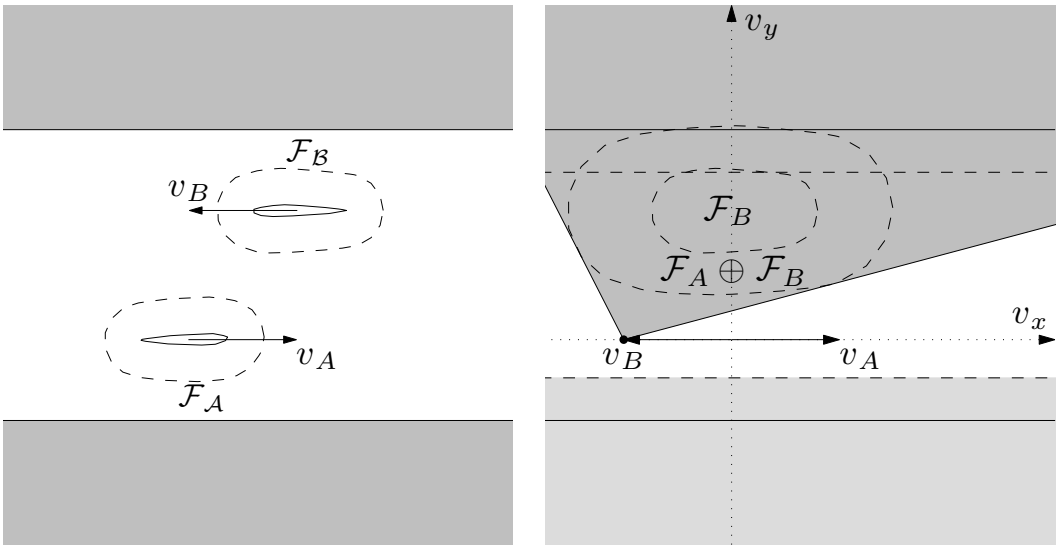
localisation. The system builds upon two main components introduced in Chapter 2, i.e. the Velocity Obstacle (VO) paradigm and AMCL, to compute collision free motions in a real-world system of robots.

While actions are computed independently for each robot, information about position and velocity is shared using local inter-robot communication. This keeps the communication overhead limited while avoiding problems like robot-robot detection. CALU uses non-holonomic optimal reciprocal collision avoidance (NH-ORCA) (Alonso-Mora et al., 2010) to compute collision free velocities and assumes disc-shaped robots with kinematic constraints. Uncertainty in localisation is addressed by inflating the robots' circumscribed radii according to the particle distribution of AMCL. For more information we refer to (Hennes et al., 2012).

As such, CALU provides a solution that is situated in-between centralised motion planning and communication-free individual navigation.

While CALU effectively alleviates the need for global positioning by using onboard localisation, some problems remain. Sub-optimal behaviour is encountered when (a) the footprint of the robot is not efficiently approximated by a disk; and (b) the pose belief distribution of AMCL is not circular but elongated along one axis (typically observed in long corridors). In both situations, the resulting VOs largely overestimate the unsafe velocity regions. Hence, this conservative approximation might lead to a sub-optimal - or no solution at all as illustrated in Figure 3.1.

As an extension, we have introduced *Convex Outline Collision Avoidance under Localization Uncertainty (COCALU)* to address these shortcomings (Claes et al., 2012). COCALU uses the same general approach as CALU, i.e. it is based on decentralised



(a) Configuration space (b) Velocity space

FIGURE 3.2: Using COCALU solves the corridor problem. Since the robots' footprints and localisation uncertainty are approximated with less overestimation, the robots can pass along the corridor without a problem.

computation, onboard localisation and local communication to share relevant shape, position and velocity data between robots.

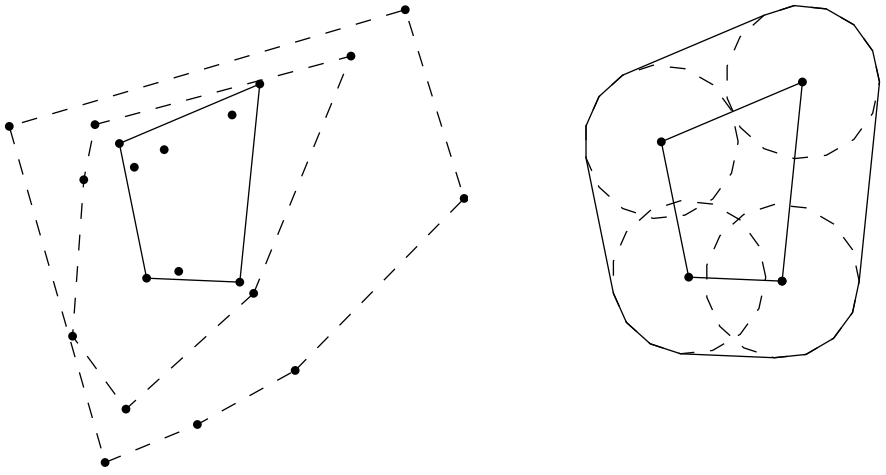
The key difference between CALU and COCALU is to use the shape of the particle cloud instead of using a circumscribed circle and the actual robot's footprint.

In this approach, we approximate the shape of the distribution of particles in AMCL by a convex hull. However, using the convex hull of all particles can results in large over-estimations, since outliers in the particles' positions inflate the resulting convex hull immensely. As a solution to this problem, we use *convex hull peeling*, which is also known as *onion peeling* (Chazelle, 1985), in combination with an error bound.

The idea behind the *onion peeling* is to create layers of convex hulls. This can be intuitively explained by removing the points on the outer convex hull, and to calculate a new convex hull of the remaining points. This process can be repeated iteratively until the remaining points are less than two. Figure 3.3a shows three iterations of the method on an example point cloud.

The general idea is that COCALU finds the convex hull layer in which the probability of the robot being located in is greater than $1-\varepsilon$, where ε is a parameter of the algorithm.

Algorithm 5 summarises the approach. It takes the weighted particle cloud from AMCL (cf. Equation 2.23) as input. As long as the sum of the weights of the removed samples, does not exceed the error bound, we create the convex hull of all (remaining) particle samples. Afterwards, we sum up all the weights of the particles located on the convex hull and add this weight to the previously computed sum. If the total sum does



(a) Convex hull peeling (b) Minkowski sum

FIGURE 3.3: (a) Three iterations of convex hull peeling. (b) Minkowski sum of the resulting convex polygon and a circular footprint.

not exceed the error bound, all the particles that define the current convex hull will be removed from the particle set and the process is repeated. When this convex hull is found, we calculate the Minkowski sum of the robot's footprint and the convex hull. The convex hull of the Minkowski sum is then used as new footprint of the robot as shown in Figure 3.3b.

Given enough samples, we can guarantee that the robot is located within the the area spanned by the remaining convex hull with probability $1 - \varepsilon$ (Claes et al., 2012).

Proof. To derive this guarantee, we revisit the particle filter described in Section 2.3.1. Let $\mathbf{x}_k = (x, y, \theta)$ be the state of the system. The posterior filtered density distribution $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ can be approximated as:

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) \approx \sum_{i=1}^{N} w_k^i \; \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right), \tag{3.1}$$

where $\delta(\cdot)$ is the Dirac delta measure. We recall that a particle state at time k is captured by $\mathbf{s}_k^i = (\hat{x}_k^i, \hat{y}_k^i, \hat{\theta}_k^i)$. In the limit $(N \to \infty)$, Equation 3.1 approaches the real posterior density distribution. We can define the mean $\mu = (\mu_x, \mu_y, \mu_\theta)$ of the distribution accordingly:

$$\begin{aligned}
\mu_x &= \sum_i w_k^i \; \hat{x}_k^i, \\
\mu_y &= \sum_i w_k^i \; \hat{y}_k^i, \\
\mu_\theta &= \text{atan2}\left(\sum_i w_k^i \; sin(\hat{\theta}_k^i), \sum_i w_k^i \; cos(\hat{\theta}_k^i)\right).
\end{aligned}$$

Algorithm 5: COCALU

Input :
$(\mathcal{F}, p, v)$: Robot footprint, position and velocity
$(s^i, w^i) \in \mathcal{P} = \mathcal{S} \times \mathcal{W}$: AMCL weighted particle set
$(\mathcal{F}_j, p_j, v_j) \in \mathcal{A}$: List of neighbouring Agents
ε : error bound
v^{pref} : preferred Velocity
τ : truncation time-steps

Output: v^{new}: New collision free velocity

$bound \leftarrow 0$
while $bound \leq \varepsilon$ **do**
  Create convex hull $\mathcal{C}$ of $\mathcal{S}$
  $bound \leftarrow bound + \sum_{\forall i: s^i \in \mathcal{C}} w_i$
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{(s^i, w^i) \in \mathcal{P} | s^i \in \mathcal{C}\}$
$\mathcal{M} \leftarrow \mathcal{F} \oplus \mathcal{C}$
foreach $(\mathcal{F}_j, p_j, v_j) = A_j \in \mathcal{A}$ **do**
  $\mathcal{M}_{A_j} \leftarrow \mathcal{F}_j \oplus \mathcal{M}$
  Construct VO_{A_j} from $\mathcal{M}_{A_j}$ at $p_j - p$
  Construct $HRVO_{A_j}$ from VO_{A_j} with v_j and v
  Construct $HRVO^{\tau}_{A_j}$ from $HRVO_{A_j}$ with τ
Use ClearPath to calculate new velocity v^{new} from v^{pref} and all $HRVO^{\tau}_{A_j}$

The mean gives the current position estimate of the robot. The probability of the robot actually residing within a certain area $\mathcal{A}$ at time k is:

$$p(\mathbf{x}_k \in \mathcal{A} | \mathbf{z_{1:k}}) = \int_{\mathcal{A}} p(\mathbf{x} | \mathbf{z_{1:k}}) d\mathbf{x}. \tag{3.2}$$

We can rewrite (3.2) using (3.1) as follows:

$$p(\mathbf{x}_k \in \mathcal{A} | \mathbf{z_{1:k}}) \approx \sum_{\forall_i : \mathbf{s}^i_k \in \mathcal{A}} w^i_k \, \delta\left(\mathbf{x}_k - \mathbf{s}^i_k\right). \tag{3.3}$$

From (3.3) we see that for any given $\varepsilon \in [0, 1)$ there is an $\mathcal{A}$ such that:

$$p(\mathbf{x}_k \in \mathcal{A} | \mathbf{z_{1:k}}) \geq 1 - \varepsilon. \tag{3.4}$$

□

To summarise, using convex hull peeling for approximating localisation uncertainty and convex footprints solves the corridor problem. Comparing Figure 3.1 and 3.2 shows the differences when using CALU and COCALU. In the latter figure, it can be seen that the robots can easily pass each other even without adapting their path.

3.3 Towards Human-Safe Pro-Active Collision Avoidance

While the previous algorithms, CALU and COCALU, provide guaranteed safety and even optimality for the individual agents, there are still some limitations that remain. Specifically, the algorithms calculate the velocity that is closest to the preferred velocity and still safe. This implies that the robots always pass each other within only marginal distances. While this approach is feasible in simulation, in real world applications it is not always possible to exactly control the velocity of the robots. With only marginal distances between the robots that pass each other, there is an increased risk that the smallest error in control will lead to a collision. An additional limitation is that all agents, either human or robot, are treated in the same way, while it would be desirable to preserve more distance from humans than from other robots.

Furthermore, if a robot knows that another robot is running the same algorithm (e.g. by using communication), it can drive closer to that robot since it can assume that the other robot will partly take avoiding actions as well. While when driving towards other robots and, particularly in the presence of humans, more distance is recommended.

To tackle these problems, we introduce a pro-active local collision avoidance system for MRS in a shared workspace that aims to overcome the stated limitations. The robots use the velocity obstacle paradigm to choose their velocities in the input space; however, instead of choosing only the closest velocity to the preferred velocity, more cost features are introduced in order to evaluate which one is the best velocity to choose. This allows us to apply different weights or importance factors for passing humans, other robots, and static obstacles. Furthermore, we introduce a smart sampling technique that limits the need to sample throughout the whole velocity space.

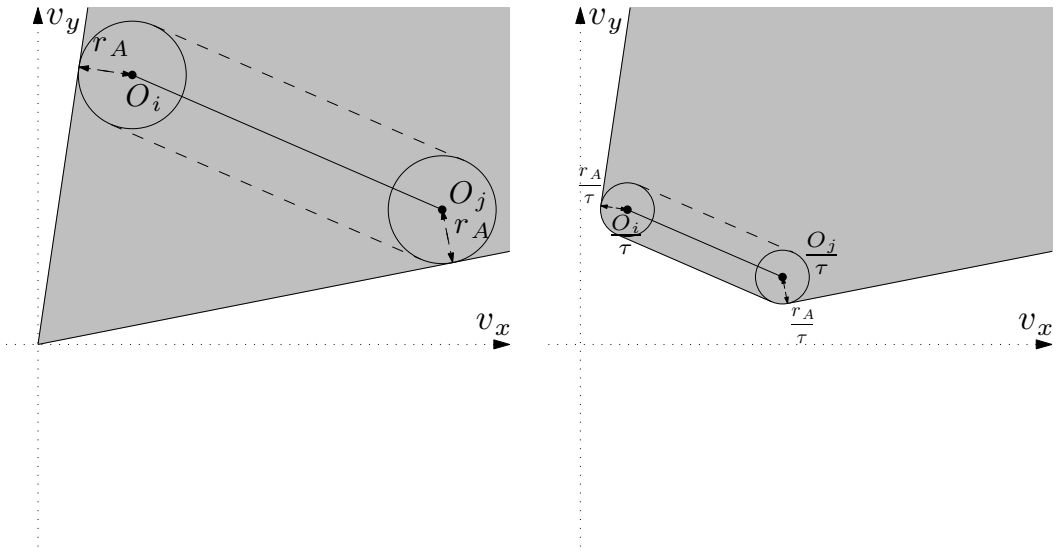
The resulting algorithm is decentralised with low computational complexity, such that the calculations can be performed online in real time, even on lower-end onboard computers.

3.3.1 Problems of Convex Outline Collision Avoidance under Localization Uncertainty

As explained previously, some problems and limitations remain when using COCALU. In the previous approach, we have focused on the robot-robot collision avoidance, i.e. the robots head with a straight path to the goal, and only needed to deviate to avoid other robots. Thus, static obstacles have been ignored.

Additionally, VO-based methods tend to end up in dead-lock situations. This means that they come to a situation in which the optimal velocity is zero, since it is the only velocity not leading to a collision. This is especially problematic with many static obstacles since the environment does not change. Thus as soon as the robot is in a situation in which the best velocity is zero it will stay this way forever.

Unfortunately, optimality can be defined in many ways. In the case of COCALU, optimality means driving as close as possible to the desired speed without collisions. In



(a) VO_{obst} for a robot R_A (b) VO^{τ}_{obst}

FIGURE 3.4: (a) Constructing a VO out of a robots' footprint and an obstacle line-segment. (b) Truncating the VO by τ.

many cases this implies that robots using this algorithm pass each other close to zero distances, i.e. there is no margin for error. In real life, where control of the robot is not instantaneous and perfectly accurate, it is likely this will lead to collisions. While COCALU implicitly provides safety by enlarging the robots' footprints by the localisation uncertainty, this is not an optimal solution. This is evident when the localisation accuracy is high, the point cloud converges to the actual robots position and the safety region decreases. Therefore, we need to explicitly take this into account.

Another limitation of the approach is that it is perceived as uncomfortable or unsafe by humans when the robots pass unnecessarily close by. An intrusion of ones personal space is usually not appreciated, especially when it concerns a robot.

In the following, we present additions and extensions to the previous COCALU approach in order to tackle the problems outlined above.

3.3.2 Static Obstacles with Velocity Obstacle-based Methods

In order to avoid static obstacles in VO-based methods they can be integrated as if they are static agents. Figure 3.4a shows the construction of a VO for a round robot with radius r_A and an obstacle line-segment defined by two points O_i and O_j. The construction follows the same rules as already presented in Section 2.1.1. Additionally, if we detect a complete outline of the obstacles, we can use the Minkowski sum of the robots footprint with the detected outline and compute the VO according to Equations 2.2 and 2.3.

Since static obstacles, by definition, do not move, we have to truncate the VO by τ, since otherwise the apex of the VO is at the origin of the velocity space, and the robot is rendered immobile as soon as it is surrounded by obstacles for example in a room.

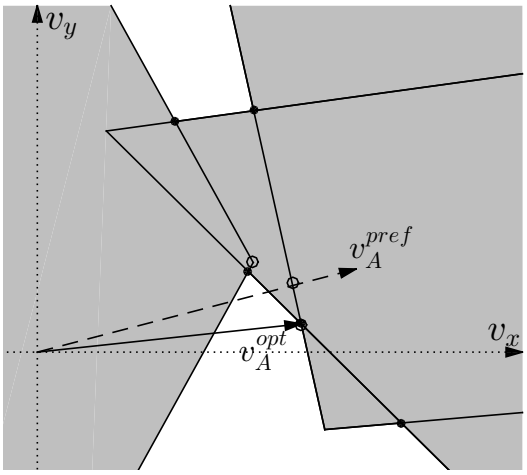


FIGURE 3.5: Increasing the footprint of the other robots is one way to create more safety. However this also reduces the available safe velocities to choose from and could lead to potential problems in dense configurations, where the whole velocity space becomes unavailable.

The walls would induce a VO in any direction, since all velocities will lead to a collision eventually.

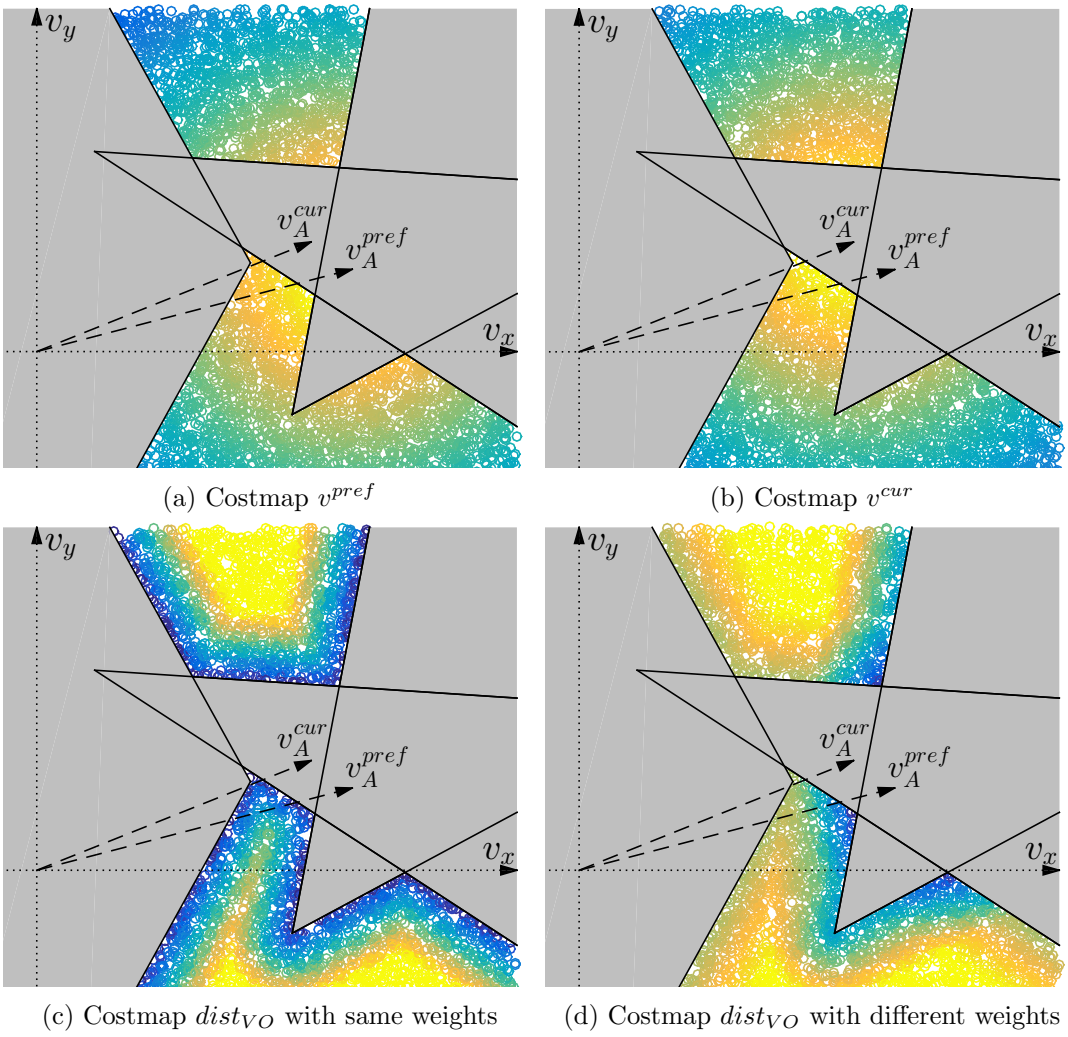
Likewise, we cannot translate the VO, e.g. to create a Reciprocal Velocity Obstacle (RVO) or HRVO, since these types of VOs are based on the assumption that the other robot takes part in the collision avoidance, which is not the case for static obstacles.

Finally, as explained above, VO-based methods tend to end up in dead-lock situations. This can be overcome by adding a global planning method on top of the local VO-based controller. The global planner computes a path to the goal, and feeds waypoints to the controller. The direction of these waypoints than determines the preferred velocity for the VO-based algorithm. As soon as the controller does not find a valid non-zero velocity, the global planner is called again in order to recompute a new path.

3.3.3 Convex Outline Collision Avoidance under Localization Uncertainty with Monte Carlo sampling

Our proposed algorithm has the same assumptions as COCALU. The robots have to be able to sense velocity and shape of other robots and humans. The detection of other robots and humans is a whole research field in itself. For instance, the *Social situation-aware perception and action for cognitive robots (SPENCER)* project [1] is a European Union funded initiative of six universities with the goal to enable robots to work in human environments. Implementing this detection based on sensors is out of scope of this thesis, thus, we rely on communication between the robots. More specifically, the robots use the same global reference frame and constantly broadcast their positions via WiFi. For the human detection, we rely on the code that was made available for ROS in the SPENCER project (Linder et al., 2016).

[1] http://www.spencer.eu/



(a) Costmap v^{pref}

(b) Costmap v^{cur}

(c) Costmap $dist_{VO}$ with same weights

(d) Costmap $dist_{VO}$ with different weights

FIGURE 3.6: Different cost functions for evaluating a velocity. Parts in yellow depict lower, i.e. better costs, and parts in blue show higher costs. The distances to the preferred velocity (a) and current velocity (b) are shown as cost, where further away yields higher cost. (c) and (d) show the distances to the VOs as costmaps, where points closer to the VOs yield higher cost.

There are multiple ways to ensure that the robots are passing each other with more distance between them. One straightforward idea is to virtually increase the size of the robots' footprints. This results in larger velocity obstacles and consequently the robots will have more distance between one another. However, this also drastically reduces the safe velocity space as shown in Figure 3.5. This approach marks more regions in the velocity unsafe and therefore reduces the options to choose from. It can lead to problems in dense situations when many other robots are present and the entire velocity space is marked unsafe though it still would be possible to manoeuvre without collisions.

To overcome this problem, we use a Monte Carlo sampling based approach with multiple cost functions. This means that the chosen velocities get evaluated not only by their distances from the preferred goal velocity but by multiple other evaluation functions. Figure 3.6 shows the result of different evaluation functions in the example

setting. The distances of the sampled velocity against the preferred velocity but also against the current velocity are shown. Likewise, it is shown how the closest distance to any velocity obstacle can be modelled as negative cost. The resulting distance can be limited, i.e. that points which are further away than a set distance do not get scored higher. This can effectively control the behaviour of the robot. Similarly, if we assume that a velocity obstacle is induced by a human, this can be weighted differently than the distances from the other velocity obstacles. The effect is shown in Figures 3.6c and 3.6d, where the right most velocity obstacle is weighted with double the cost than the other two velocity obstacles. Using this approach, we can model the personal space of a human by setting the cost for intrusion very high up to a certain distance. For personal space, a distance of 50 cm is usually regarded as applicable (Kruse et al., 2013).

In order to select the optimal velocity, we sample inside the velocity space and translate the velocity to non-holonomic motions afterwards. A velocity sample that points inside a VO is disregarded since it is unsafe. Figure 3.7 shows the costmaps and the resulting optimal velocity. As can be seen in Figure 3.7a, the resulting velocity is close to the originally calculated optimal velocity when using ClearPath. However, when the VOs are weighted differently, the optimal velocity is in a different region of the velocity space as shown in Figure 3.7b.

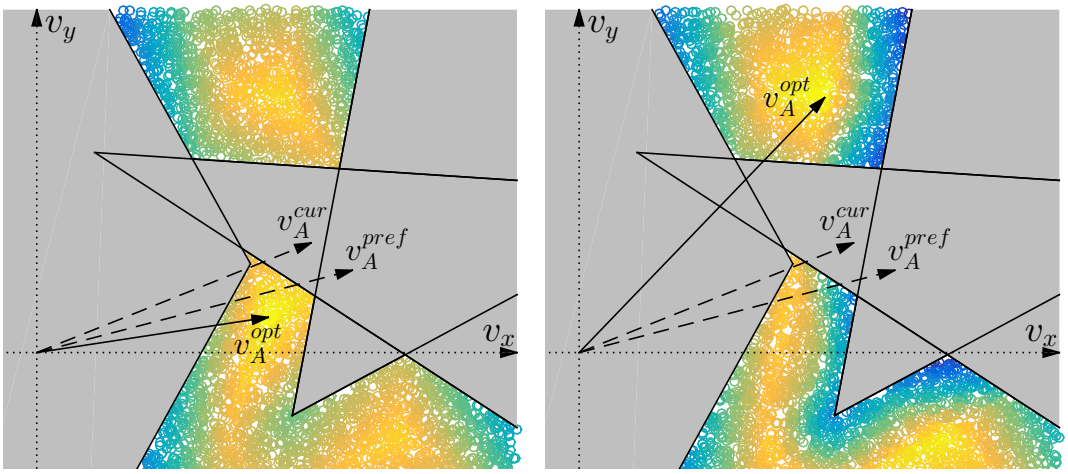
We can combine the ClearPath algorithm with the above idea to incorporate a smarter sampling algorithm. The ranked velocities calculated by ClearPath are used as a seed (see Figure 2.5a: points marked as v^{opt_i}), such that samples are only created in the vicinity of these velocities. Figure 3.8 shows the idea of this algorithm. The trade-off of this approach is that it might miss the global optimum in favour of being computationally faster.

Lastly, we can also adapt the truncation factor to improve the safety against other uncontrolled robots and humans. A higher truncation time results in safer velocities, since it determines the time the chosen velocity is guaranteed to be collision-free in the current configuration of the system.

3.3.4 Convex Outline Collision Avoidance under Localization Uncertainty with the Dynamic Window Approach

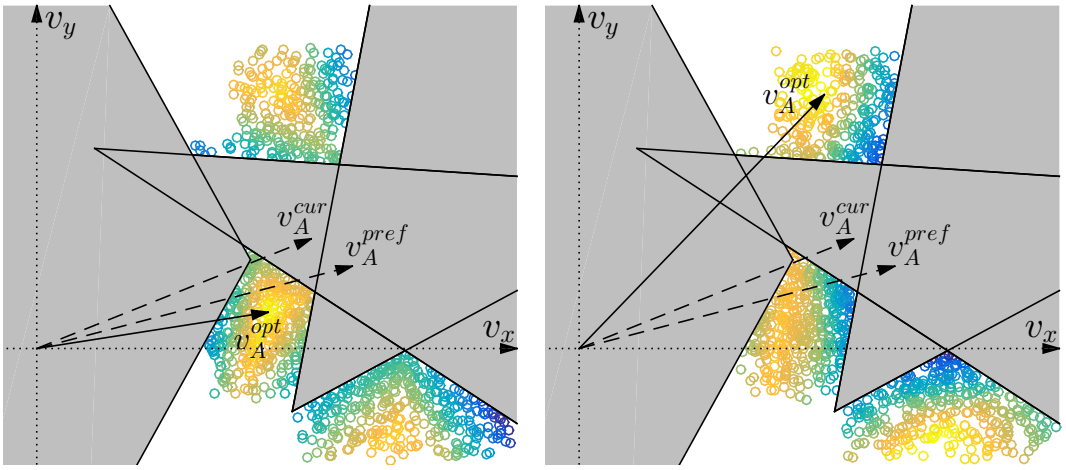
In the previous section, we have shown on how to use Monte Carlo simulations to generate the samples for the velocity of the robots. As another solution, we can generate the samples according to the motion model of the robots and translate the velocities based on the motion model to an approximated holonomic speed. This idea of so-called trajectory rollouts is applied in the well know DWA (Fox et al., 1997) which is commonly used in ROS.

The controller generates velocities according to the dynamic motion constraints of the robots and predicts the position-based motion model of robots. Each trajectory is evaluated according to various cost functions as presented in Section 2.3.2. While these cost function are in configuration space, and not in velocity space, the similarities



(a) v^{opt} with all VOs weighted equally (b) v^{opt} with different VO weights

FIGURE 3.7: Selecting the optimal velocity based on different combinations of the costmaps and sampling throughout the full velocity space. (a) All VOs are weighted equally. (b) The VO on the right has additional weight.



(a) v^{opt} with same VO weights and smart sampling (b) v^{opt} with different VO weights and smart sampling

FIGURE 3.8: Applying smart sampling only around the best v_{cp}^{opt} points as calculated by ClearPath. (a) All VOs are weighted equally. (b) The VO on the right has additional weight.

to our COCALU with Monte Carlo sampling approach allows us to easily combine the two planners. We can use DWA to generate the velocity samples and trajectories and evaluate the configuration space based critics as the normal DWA would use and then evaluate the trajectory based on our COCALU with Monte Carlo sampling cost functions. Since the trajectory is already available, the translation to velocity space is straight forward, by computing the differences of the starting point and end points and dividing by the simulation time.

A major advantage is that, since DWA is already commonly used, the COCALU cost functions can easily be added to any robot that is using DWA.

As previously mentioned, a common problem with VO-based approaches, is that the velocity space is too restricted when also including static obstacles, even when truncating the VO. Furthermore, as static obstacles are immobile, it is preferable to deal with them in the configuration space. When using a trajectory rollouts approach as with DWA, we can check collisions with static obstacles by imposing our footprint on the resulting trajectory. If the footprint collides with any static obstacle, the trajectory is invalidated and discarded for this iteration.

Thus we can combine the DWA cost functions in the configuration space for dealing with features that are well represented in that space (e.g. collisions with static obstacles, progress towards the goal), with the COCALU cost functions, which are well suited to avoid dynamic obstacles. As a result, we have a navigation approach that is highly flexibly and can be used in various environments.

3.3.5 Pro-Active Collision Avoidance

An advantage when using any velocity obstacle based approach is that we can easily have pro-active collision avoidance, even when the robots are standing still. When not moving, the robots' preferred velocity is zero, which can be evaluated using the same approach as while driving. Thus, when remaining at the same position would result in a collision, the robots using this approach will pro-actively take actions and avoid the incoming robot or human. This is of course only necessary when the incoming robot is not already taking care of the avoidance itself. In the latter case, i.e. if the stationary robots know (for instance, by using communication) that the incoming robot is already taking care of the collision, the robots that are standing still will detect that their preferred velocity, i.e. zero, does not lead to collision again, thus they remain in-place.

3.3.6 Complexity Analysis of the Approach

The complexity of this approach is the time needed to generate and evaluate all samples S. For the evaluation, the distances to any approximated truncated VO, and the distances to the preferred and current velocity have to be calculated. These are all geometric operations with linear complexity. Thus the evaluation of the samples runs in $\mathcal{O}(S \times N))$, where N is the number of neighbouring robots. The generation of the samples when sampling in the velocity space is only depending on the number of samples and the random generator used. When we use the motion model to generate our samples, we can pre-compute the motions and use these motion primitives in a lookup, so the sample generation is $\mathcal{O}(S)$. If we use the dynamic window approach, we need to recompute the samples according to the current state of the robot. This depends on the resolution of the trajectory-rollouts. As previously stated, ClearPath runs in $\mathcal{O}(N(N+M))$, where M is the number of total intersection segments (Guy et al., 2009).

3.4 Experiments and Results

The presented algorithms are implemented in the open source *ROS* (Quigley et al., 2009) framework as presented in Section 2.3. The code for the implementation in ROS can be found on github[2]. As described earlier, we rely on communication between the robots to broadcast their positions in a common reference frame. The robots are controlled at 10 Hz, and at each time-step the robots evaluate the current position and independently choose their preferred velocity.

As baselines, we use the original $COCALU$ approach and also the commonly used DWA method. These baselines are compared with the newly proposed $COCALU$ with Monte Carlo Sampling using the smart sampling as explained in Section 3.3.3 referred to as $COCALU^{sampling}$, and with the approach that combines $COCALU$ and DWA as explained in Section 3.3.4, to which we refer to as $COCALU^{dwa}$.

We evaluate several performance measures: a) number of collisions and deadlocks, b) time to complete a single run, c) distance travelled and d) jerk cost. The jerk cost measures the smoothness of a path. More specifically, the jerk is the change in acceleration over time. It is defined as:

$$Jerk_{lin} = \frac{1}{2} \int \dddot{\mathbf{x}}(t)\ dt \tag{3.5}$$

$$Jerk_{ang} = \frac{1}{2} \int \dddot{\theta}(t)\ dt \tag{3.6}$$

where $\mathbf{x}$ is the forward displacement of the robot, i.e. the linear speed is $\dot{\mathbf{x}}$ and θ the robot's heading, i.e. $\dot{\theta}$ is the angular speed. A deadlock is defined in this case when the goals are not reached within 60 seconds and there is no collision present. Runs in which collisions occurred or which had deadlocks, are excluded from the averages of the jerk, distance and time calculations.

For all algorithms we use truncation of the velocity obstacles induced by other robots with $\tau = 10$, while for VOs induced by static obstacles we used $\tau = 1$. As static obstacles do not move, the truncation factor can be much decreased. The localisation uncertainty is set to $\epsilon = 0.3$, thus we include 70% of the particles in our footprint enlargement. This was selected by comparing the increase in footprint size against the average localisation error, such that the enlargement was enough to cover the mean localisation error.

All costmaps are included for the sampling approaches and are weighed equally. In the cases where there is a velocity obstacle induced by an uncontrolled robot or a human, the minimum distance to these velocity obstacles is weighed double.

FIGURE 3.9: A sample configuration of the simulation environment for the antipodal circle setting with eight robots.

3.4.1 Simulation Runs

We have evaluated our approach in simulation using *Stage* (Gerkey et al., 2003; Vaughan, 2008). Simulation allows us to investigate the system performance using many repetitions and various extreme settings, i.e. a very dense settings with a lot of robots.

The robots are all controlled independently and running localisation with a simulated Light Detection and Ranging (LIDAR) that is updated at 10 Hz, and has a 180 degrees field of view. Only the positions are shared via the ROS message passing system.

For the original $COCALU$ and the $COCALU^{sampling}$ approach we used the simulated LIDAR to detect the outlines of static obstacles, using the approach as described in Subsection 3.3.2. The $COCALU^{dwa}$ approach uses the configuration space costmaps to check the distances to the static obstacles during the trajectory rollouts. Also the distance to the path and the goals are scored.

A common scenario to evaluate movement in dense environments is to place a number of robots on a circle (equally spaced). The goals are located on the antipodal positions, i.e. each robot's shortest path is through the centre of the circle, see (Berg et al., 2011; Alonso-Mora et al., 2010). We use a circle with a radius of 1.7 meter in simulation. The goal is assumed to be reached when the robots centre is within a 0.15 meter radius of the true goal. This tolerance is necessary since every robot uses the AMCL for its localisation. We evaluate this scenario from 2 up to 10 robots. A sample configuration of the simulation environment is shown in Figure 3.9.

All experiments in simulation are run on a single machine with a quad-core 3.4 GHz Intel i7 processor and 16 GB of memory. Each setting is repeated 50 times and the results are averaged. As stated above, for the evaluation of jerk, distance, and travel

[2] https://github.com/daenny/collvoid

TABLE 3.1: Collisions (first number) and deadlocks (second number) for the different settings. For visual purposes, $COCALU$ is abbreviated with C.

(a) Antipodal circle

	2	3	4	5	6	7	8	9	10
DWA	50/0	50/0	50/0	50/0	50/0	50/0	50/0	50/0	50/0
C	0/0	0/0	0/0	0/2	0/0	0/2	0/4	0/3	2/2
$C^{sampling}$	0/1	0/4	0/2	0/7	0/4	0/0	0/2	0/7	0/14
C^{dwa}	0/0	0/0	0/0	3/0	4/0	10/0	13/0	11/1	14/1

(b) Random with 6 obstacles

	2	3	4	5	6	7	8	9	10
DWA	9/0	14/0	25/1	28/0	37/0	46/1	49/0	50/0	50/0
C	0/1	2/2	2/7	4/6	3/4	3/10	7/10	4/15	12/16
$C^{sampling}$	0/0	0/7	0/5	2/6	1/9	2/10	5/7	1/13	4/15
C^{dwa}	0/0	2/0	2/1	0/1	3/2	8/3	5/2	5/4	17/5

(c) Random with 10 obstacles

	2	3	4	5	6
DWA	3/1	13/3	30/2	37/3	46/2
C	1/6	2/6	2/9	5/14	4/26
$C^{sampling}$	1/5	2/9	4/7	7/19	5/15
C^{dwa}	2/2	3/1	11/1	6/6	7/6

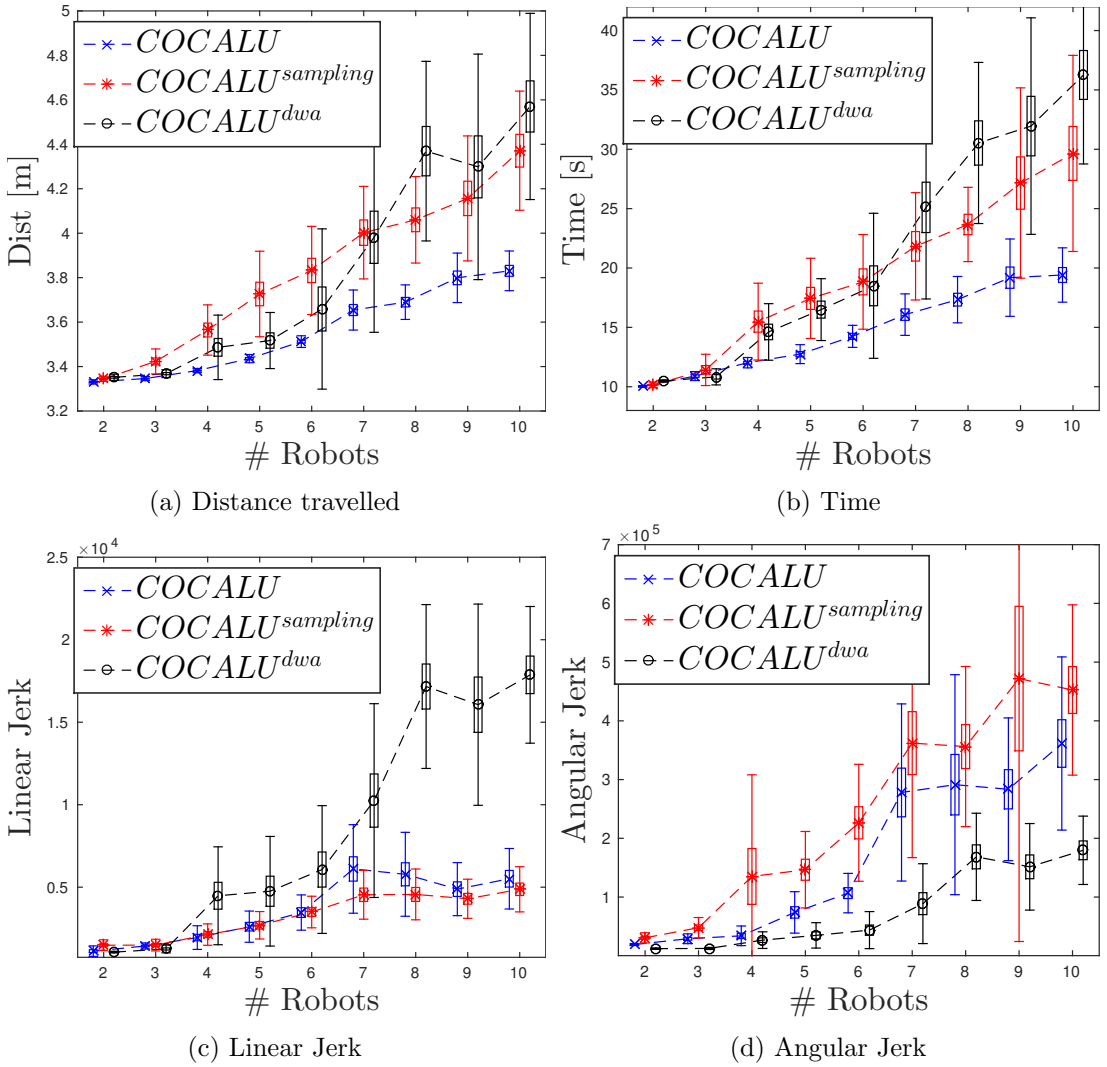
time runs in which collisions occurred or which had deadlocks, are excluded from the averages. We calculate 90% confidence intervals using the student t-distribution. The simulations are run in real time, since the message passing is an essential component of the described approach. As the ROS message passing uses real time serialisation and deserialisation, increasing the simulation speed would lead to inaccurate results.

For another less symmetric scenario, we confined the robots in a 5 meter by 5 meter square room and placed them using a uniform random distribution. Additionally, static obstacles with a square size of 0.4 meters by 0.4 meters were placed in the same environment. The generated positions were constrained such that each robot was at least 0.9 meters apart, to ensure that it is not in collision with another robot or a static obstacle.

The goals for the robots were also randomly generated, with the condition that they have to be at least two meters away from the current position. We call this the *random with obstacles* setting. This setting is evaluated with 6 static obstacles and 2 up to 10 robots, and with 10 static obstacles and 2 up to 6 robots.

The amount of collisions and deadlock are summarised in Table 3.1.

In the antipodal circle experiment (Table 3.1a), using $COCALU$ only in two runs with ten robots a collision occurred, and for $COCALU^{sampling}$ no collision occurred at all. With $COCALU^{dwa}$ the amount of collision runs increases from three with five robots up to fourteen with 10 robots. The pure DWA method, does not have any way



(a) Distance travelled

(b) Time

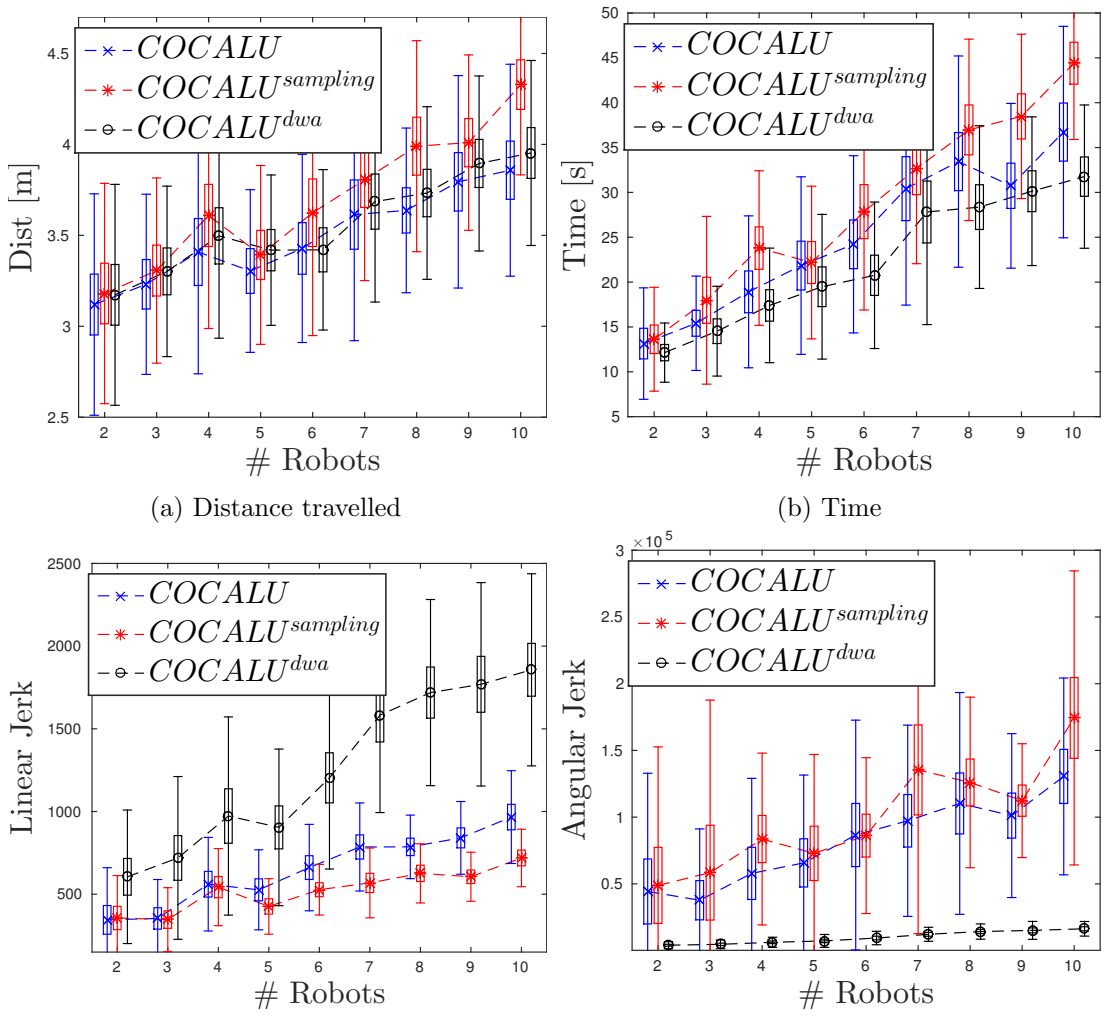
(c) Linear Jerk

(d) Angular Jerk

FIGURE 3.10: Evaluation of 50 runs in simulation of the antipodal circle experiment with 2 to 10 robots. The (a) distance travelled, (b) time to complete, and linear (c) and angular (d) jerk are shown. The boxes show the 90% confidence intervals and the whiskers show the standard deviation.

to avoid the incoming robots. As the shortest path is through the centre of the circle for every robot, the robots collide in every run.

The collisions that occur with the other approaches can have multiple reasons. As said before, the localisation uncertainty epsilon was set to 0.3, which means that there is a chance that collisions between the robots happen, when AMCL is unable to track the robots' positions sufficiently accurate. Additionally, the limited update rate of 10 Hz and the low fidelity of the simulator, which only approximates the kinematics of the robots, might lead to inaccurate trajectories and therefore collisions. Especially for *COCALU* and $COCALU^{sampling}$, collisions happened with the static obstacles, since using the LIDAR for the detection based on the outlines and then using the VO approach can lead to inaccurate footprints due to noise in the measurements. $COCALU^{dwa}$ has the advantage of dealing with the static obstacles in configuration space, which leads



(a) Distance travelled

(b) Time

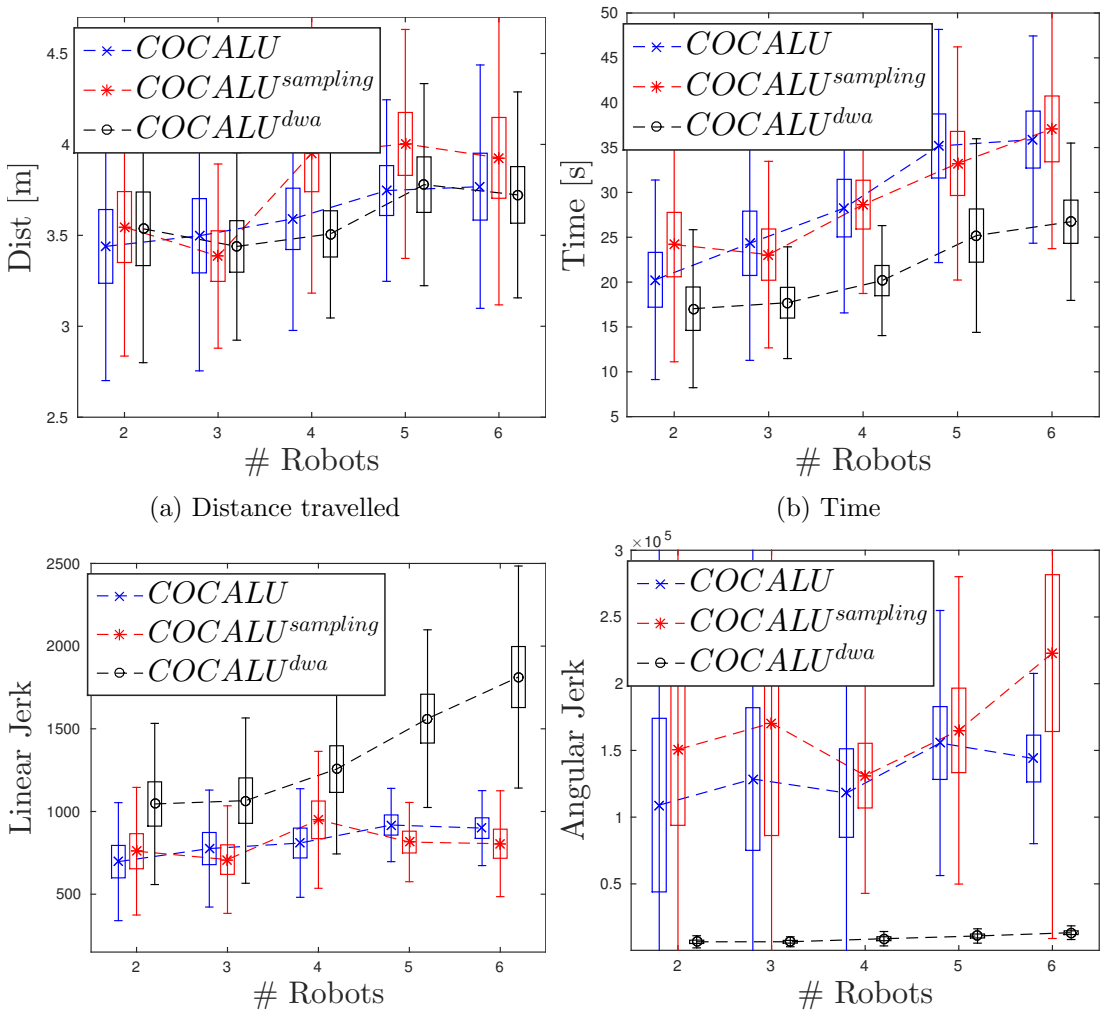
(c) Linear Jerk

(d) Angular Jerk

FIGURE 3.11: Evaluation of 50 runs in simulation of the random with 6 obstacles experiment with 2 to 10 robots. The (a) distance travelled, (b) time to complete, and linear (c) and angular (d) jerk are shown. The boxes show the 90% confidence intervals and the whiskers show the standard deviation.

to a more accurate representation and less collisions with static obstacles. However, for $COCALU^{dwa}$ the collisions usually occurred with other robots. If not configured correctly, the costs for the distance to the paths and goal can have too much weight such that the robot acts mostly according to those and does not take actions to avoid the collisions according to the $COCALU$ cost functions. At some point the robot is in a inevitable collision state, i.e. it cannot prevent collision due its kinematic constraints and the other robots are too close.

With $COCALU^{sampling}$ the amount of runs that exceeded the 60 seconds time limit increases with more and more robots. This is usually due to having a dead-lock situation. More specifically, this means that some robots already have reached their goals, while the others are trapped behind these robots and are not able to reach their goals anymore. With the sampling based method, this can happen due to the different cost-functions that incentivise safe paths, which is to stay away from the other robots.



(a) Distance travelled

(b) Time

(c) Linear Jerk

(d) Angular Jerk

FIGURE 3.12: Evaluation of 50 runs in simulation of the random with 10 obstacles experiment with 2 to 6 robots. The (a) distance travelled, (b) time to complete, and linear (c) and angular (d) jerk are shown. The boxes show the 90% confidence intervals and the whiskers show the standard deviation.

For the random setting, the results are more mixed. The original *DWA* method was able to complete a couple of runs. Especially with few, i.e. two or three, robots, there is a chance that the paths of the robots do not even cross. On the other hand, we can see that if the number of robots is increased, and the environment becomes more complex, *DWA* is not able to deal with it at all, leading to failing almost every single run due to collisions.

When comparing the *COCALU* based methods, the original *COCALU* method performs worst for most of the scenarios. With 10 obstacles, $COCALU^{dwa}$ yields the best performance, while with 6 obstacles the differences are less pronounced. This is probably due to the better handling of the obstacles when using the configuration space instead of translating the outlines of the static obstacles into VOs.

The results for the other metrics are summarised in Figures 3.10, 3.11 and 3.12. The results for *DWA* are excluded since the amount of runs with collisions was too high

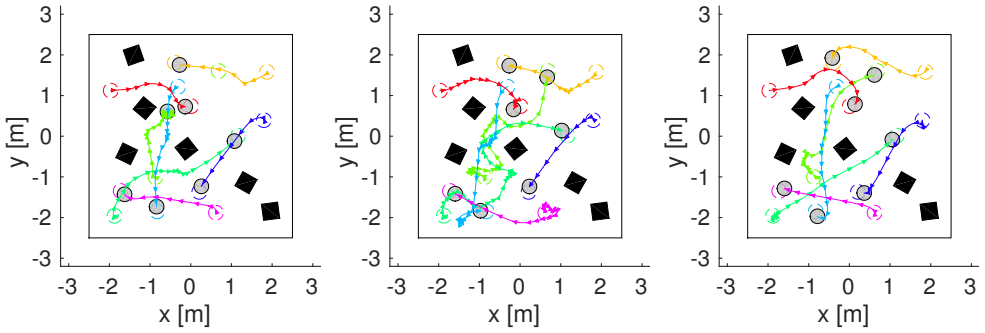
to build sensible statistics. For the antipodal circle experiment (Figure 3.10), we can see that the $COCALU^{dwa}$ approach performs comparable to the $COCALU^{sampling}$ and $COCALU$ approaches in terms of time and distances travelled. This holds for up to six robots. Afterwards, the performance deteriorates. The $COCALU^{sampling}$ approach uses more time and travels farther than the original $COCALU$ approach. This is to be expected since the robots deviate from the fastest path in order to improve safety and the costmaps are designed to give incentives to not choose the velocities which leave no margin for error.

When looking at the linear and angular jerk, it can be seen that $COCALU$ and $COCALU^{sampling}$ use significantly more angular jerk, while $COCALU^{dwa}$ uses a lot more linear jerk. This is due to the differences in the sampling methods for the velocities. $COCALU$ uses ClearPath for selecting the best velocity and $COCALU^{sampling}$ uses smart sampling around the ClearPath points to find the best velocity. These are based on the (holonomic) velocity space and then translated into linear and angular commands. Thus when the ClearPath point switched, this leads to a large change in the angular velocity, while $COCALU^{dwa}$ uses trajectory rollouts which are sampled based on the kinematic model of the robot, leading to less changes in direction, but more in linear acceleration and deceleration.

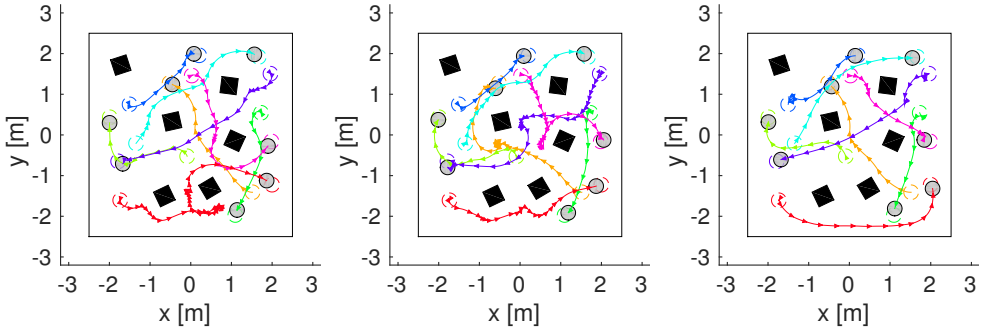
For visual inspection, some sample trajectories for 7 up to 10 robots and 6 obstacles are shown in Figure 3.13. Generally, it can be observed that $COCALU^{dwa}$ has smoother trajectories, which reflects the less usage of angular jerk. With the other two approaches, the robots manoeuvre more. Additionally, in the setting with 10 robots, it can be seen that the robot with red traces, starting in the lower right corner, with $COCALU$ has a collision with a static obstacle, while with the other two approaches, the robot reaches its goal.

Pro-Active Collision Avoidance

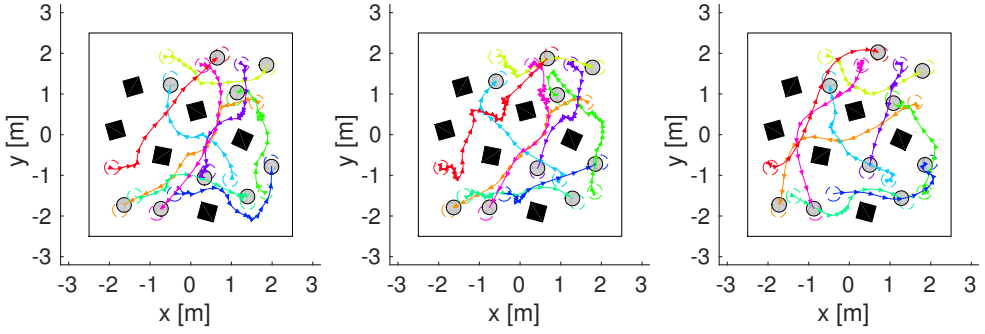
To show how the pro-active collision avoidance works, Figure 3.14 shows the trajectories of one "uncontrolled" robot passing through a crowd of robots. "Uncontrolled" in this experiment means that the robot disregards the existence of the other agents and just drives straight without taking any avoiding measures. Thus, the five robots in the centre have to pro-actively move out of the way in order to ensure safety. The robot with the blue trace is approaching, while the pink and green traced robots start moving out of the way (Figure 3.14a). As soon as the uncontrolled robot has passed, pink returns to its position (Figure 3.14b). The same happens with the green robot (Figure 3.14c). The robot with the yellow traces just moves a little bit to clear the way, however due to localisation uncertainty, the position changes such that it becomes necessary for the robot to make a more elaborate manoeuvre to reach back to its original position. The final positions and complete trajectories can be seen in Figure 3.14d.



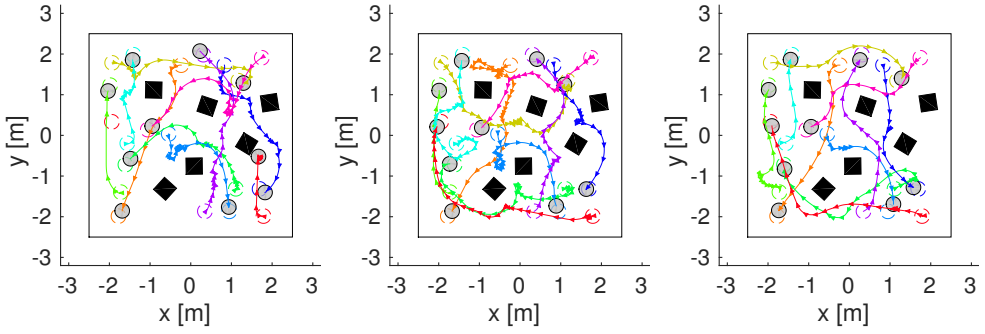
(a) 7 robots, 6 obstacles



(b) 8 robots, 6 obstacles



(c) 9 robots, 6 obstacles



(d) 10 robots, 6 obstacles

FIGURE 3.13: Sample trajectories for the setting with random goals and 6 randomly placed obstacles. Trajectories with 7 to 10 robots from top to bottom and the standard $COCALU$ (left), $COCALU^{sampling}$ (centre) and $COCALU^{dwa}$ (right). The initial and target positions are marked with dashed circles.

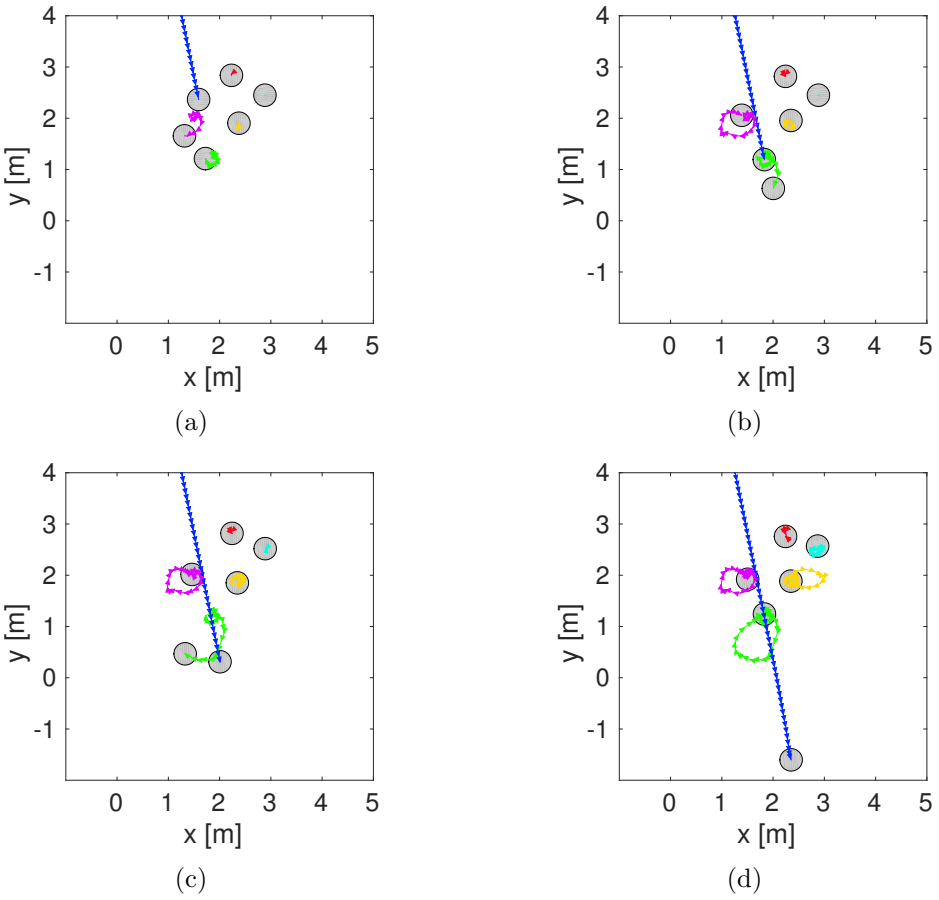
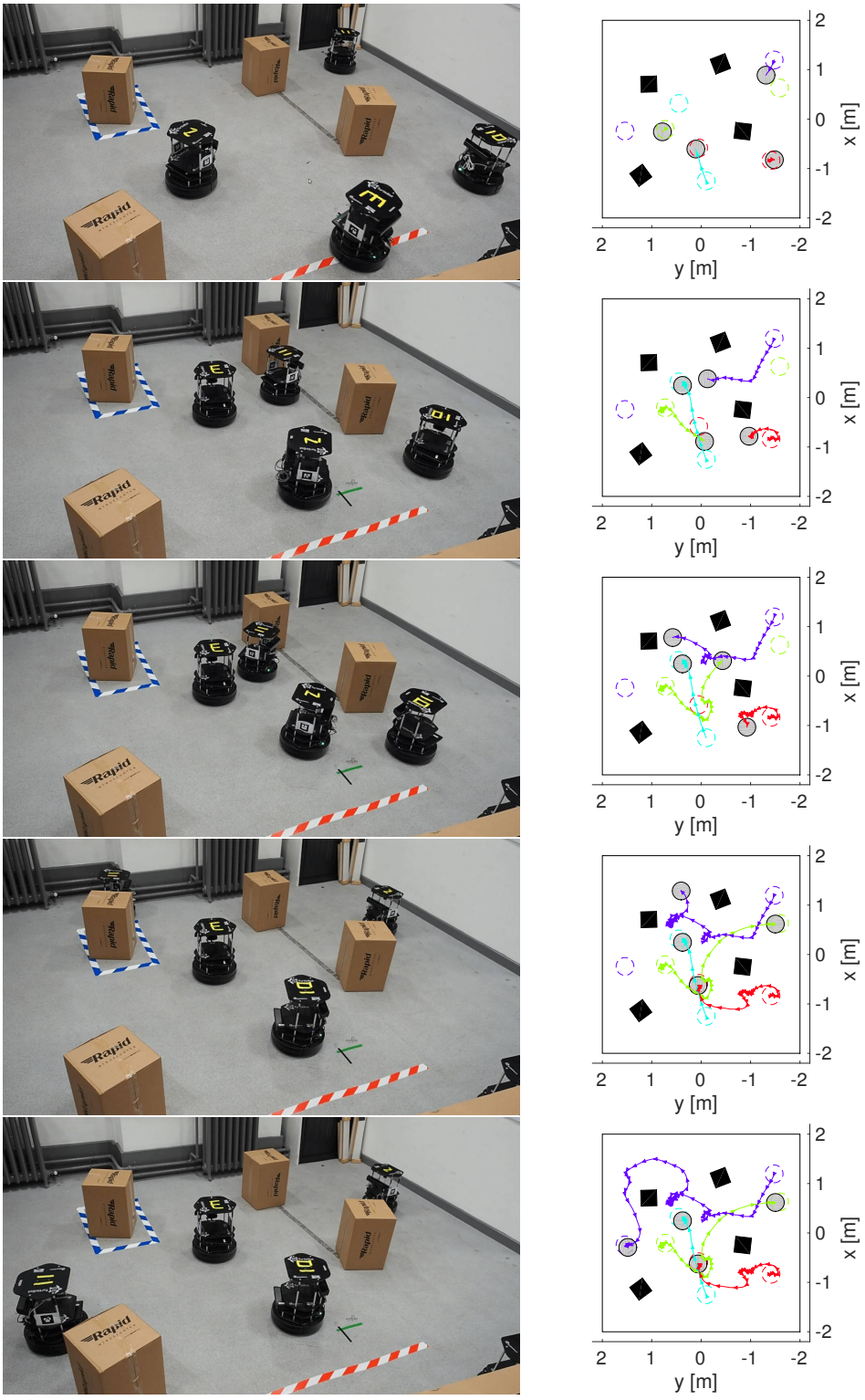


FIGURE 3.14: Pro-active collision avoidance. The *uncontrolled robot* (blue trace) neglects the presence of the other robots and drives straight towards a crowd of other robots. (a) The robot shown with pink traces is the first to start avoiding to ensure safety. (b) The robots with green and yellow traces have to move out of the way, while the pink (c) returns to its original place. (d) Due to localisation errors, the yellow traced robot has to readjust its position to return to its original place.

3.4.2 Real World Experiments

We evaluate the performance in a real-word setting using up to four differential drive Turtlebots. In addition to the usual sensors, they are equipped with a LIDAR finder to enable better localisation in larger spaces as described in Appendix A.2. All computation is performed onboard on an Intel i3 380UM 1.3 GHz dual core CPU notebook. Communication between the robots is realised via a 2.4 GHz WiFi link using a UDP connection and the LCM library (Huang et al., 2010). For human detection we use the SPENCER project code (Linder et al., 2016). It uses the laser range-finder to detect and match human legs and tracks the resulting people. We ran the random setting with four obstacles, and the antipodal circle experiment which included one human. The trajectories of the robots are recorded using the positions determined by AMCL and the human trajectories that are shown are approximated. The obstacles are shown at their estimated positions. The robots were running the $COCALU^{dwa}$ algorithm for navigation.



(a) (b)

FIGURE 3.15: A run in the real world setting. (a) Pictures of the actual run. (b) Trajectories of the robots. The initial and target positions are marked in dashed circles. The robot with purple traces (starting top right) has to readjust his path twice. The red robot waits until the green robot has passed.

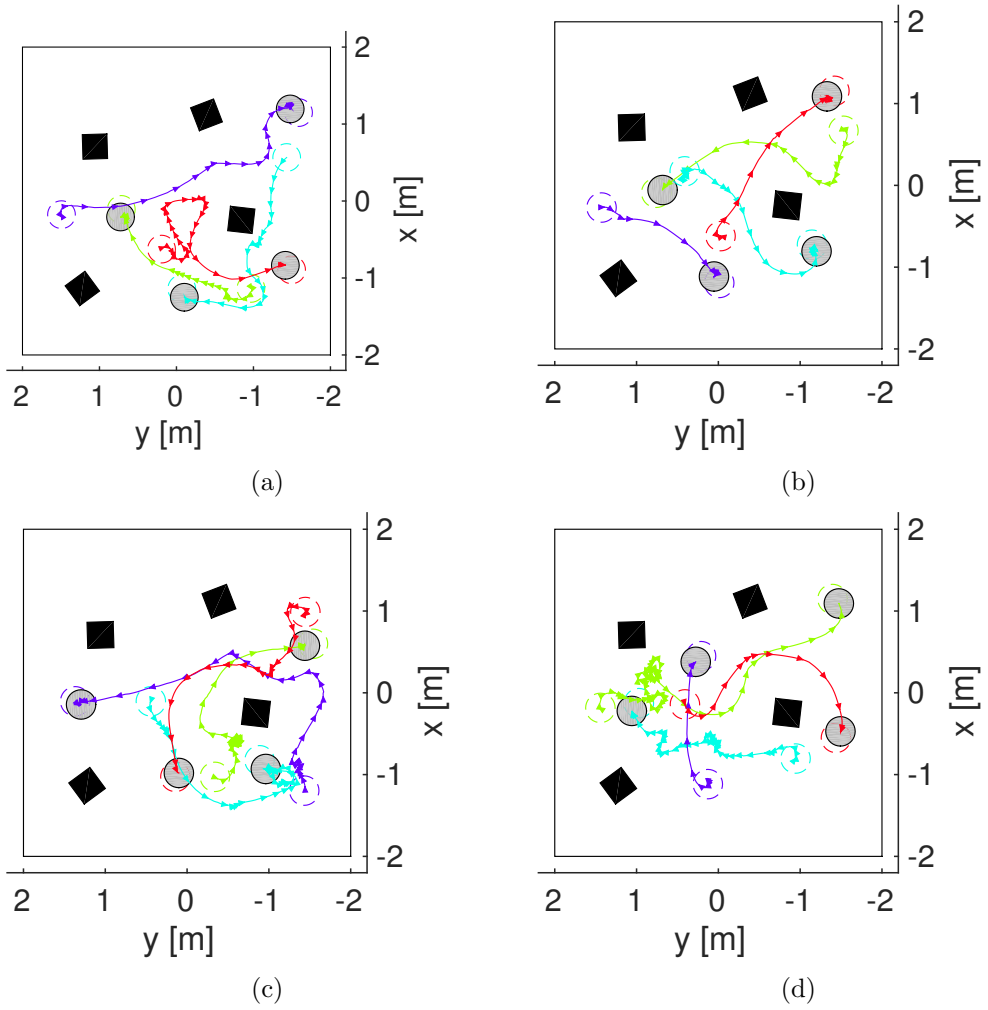
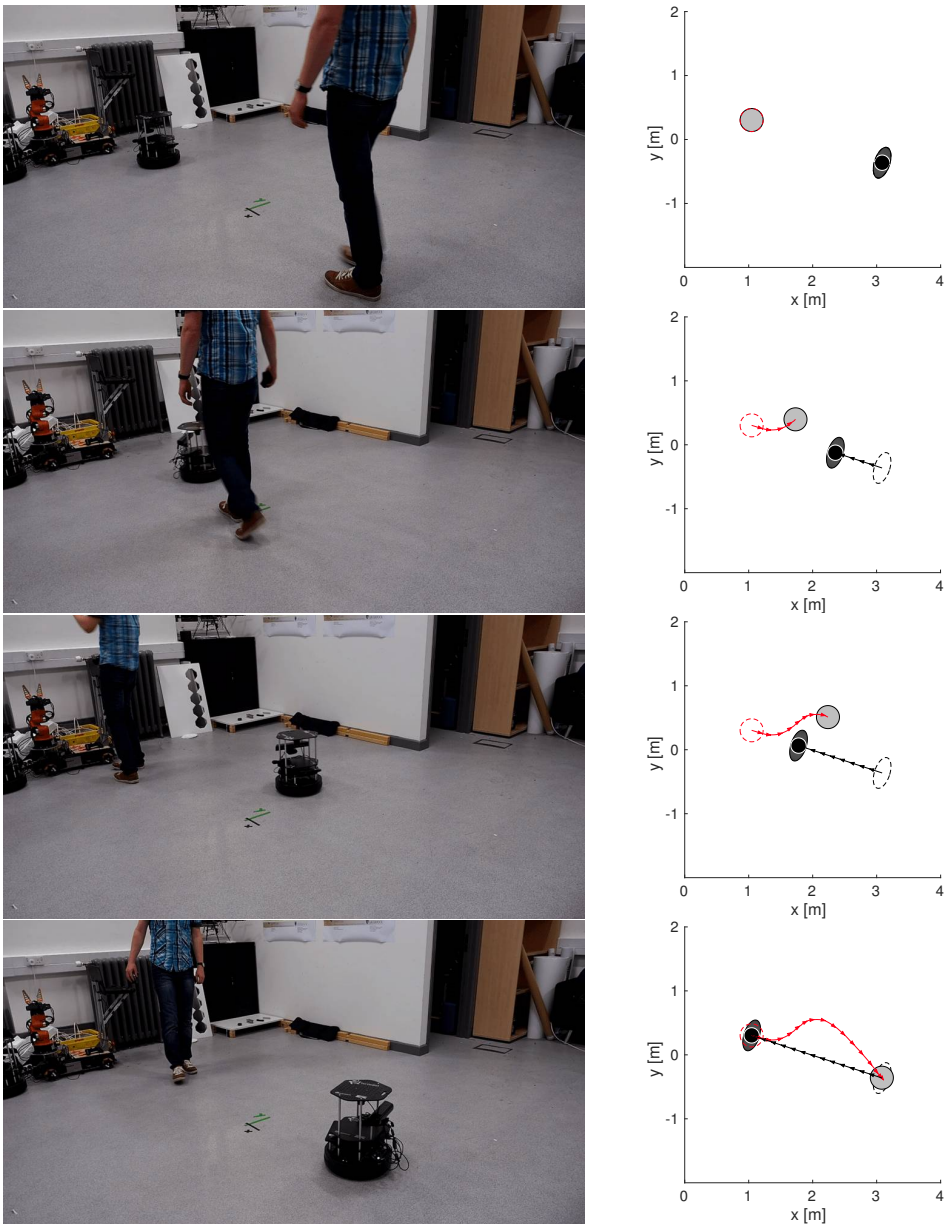


FIGURE 3.16: Several trajectories in the real world setting. The initial and target positions are marked with dashed circles.

Random with obstacles

A sample trajectory of the random with obstacles setting is shown in Figure 3.15. The left column (Figure 3.15a) shows the photos of the run, while the right column shows the trajectory plots over time (Figure 3.15b). It is tested with four Turtlebots and four obstacles. The light blue robot, starting on the bottom has a direct path to the goal. It blocks the green robot, which also reacts on the purple robot that is heading towards it. The purple robot re-plans and deviates from its original path, while the red robot waits until the green robot has passed. Afterwards the purple robot has to change its path again due to the previously unseen obstacle. Finally, all robots have reached their goals.

Some more sample trajectories of this setting are shown in Figure 3.16. From visual inspection, it can be seen that the robots drive smoothly for most of the trajectories, while for sample trajectory shown in Figure 3.16d, it shows the difficulties that $COCALU^{dwa}$ has in very dense configurations. The robot with the green traces, starting left, has to wait first for the robot with the red traces to move away. In the mean



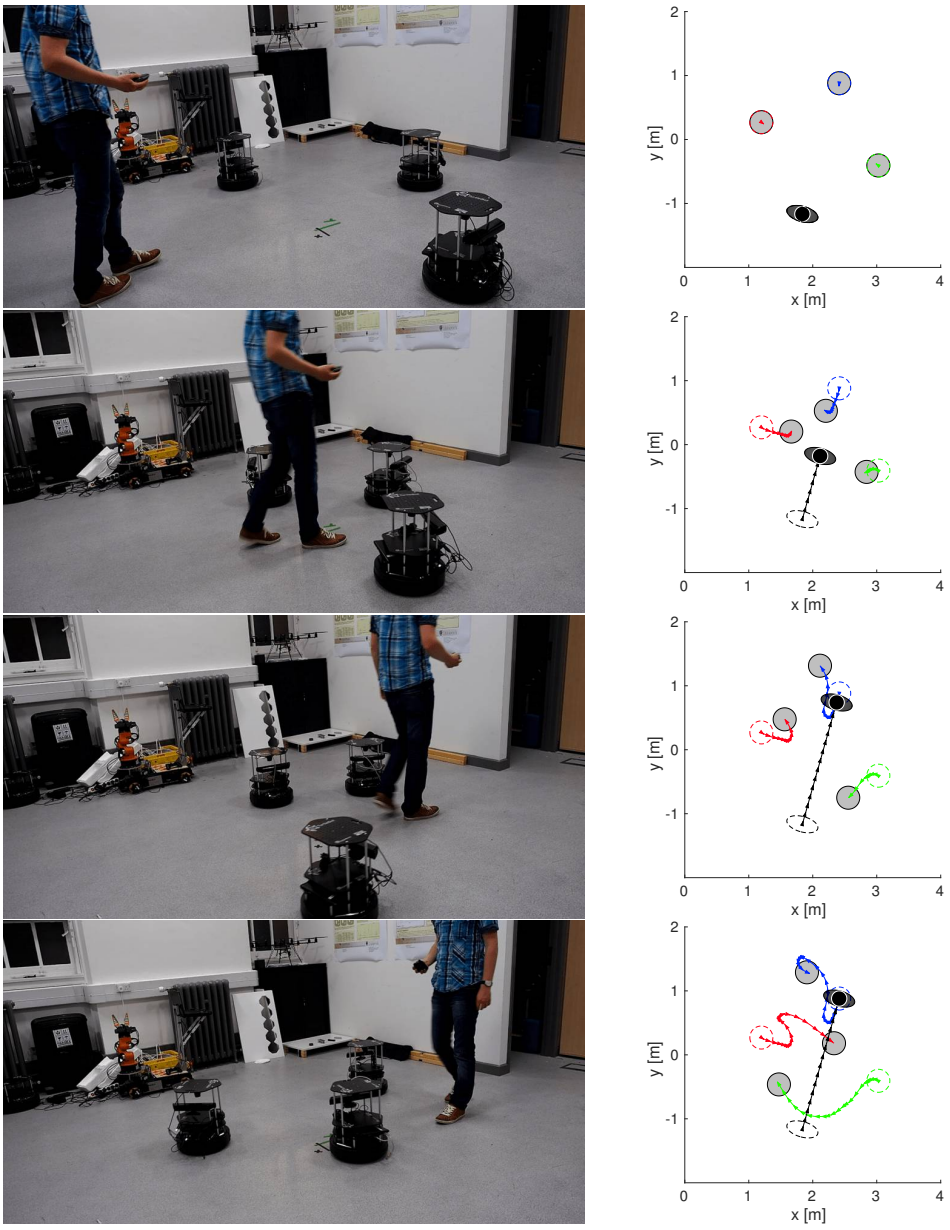
(a) (b)

FIGURE 3.17: Testing with a human with the antipodal circle experiment in the real world. A single robot switches place with a human. It detects the human and avoids him by adjusting its the path. (a) Pictures of the actual run. (b) Trajectories of the of the robot and the human. The human trajectory is approximated.

time, the other two robots (with purple and light blue traces) start moving towards their goal positions forcing the green robot to adjust its path multiple times to avoid them. Eventually, the robots are at their goal positions and the green robot can pass.

Antipodal circle

We tested our approach as well with a human switching place with a robot and passing through a crowd of robots. Pictures of the runs are shown in Figure 3.17 and Figure 3.18.

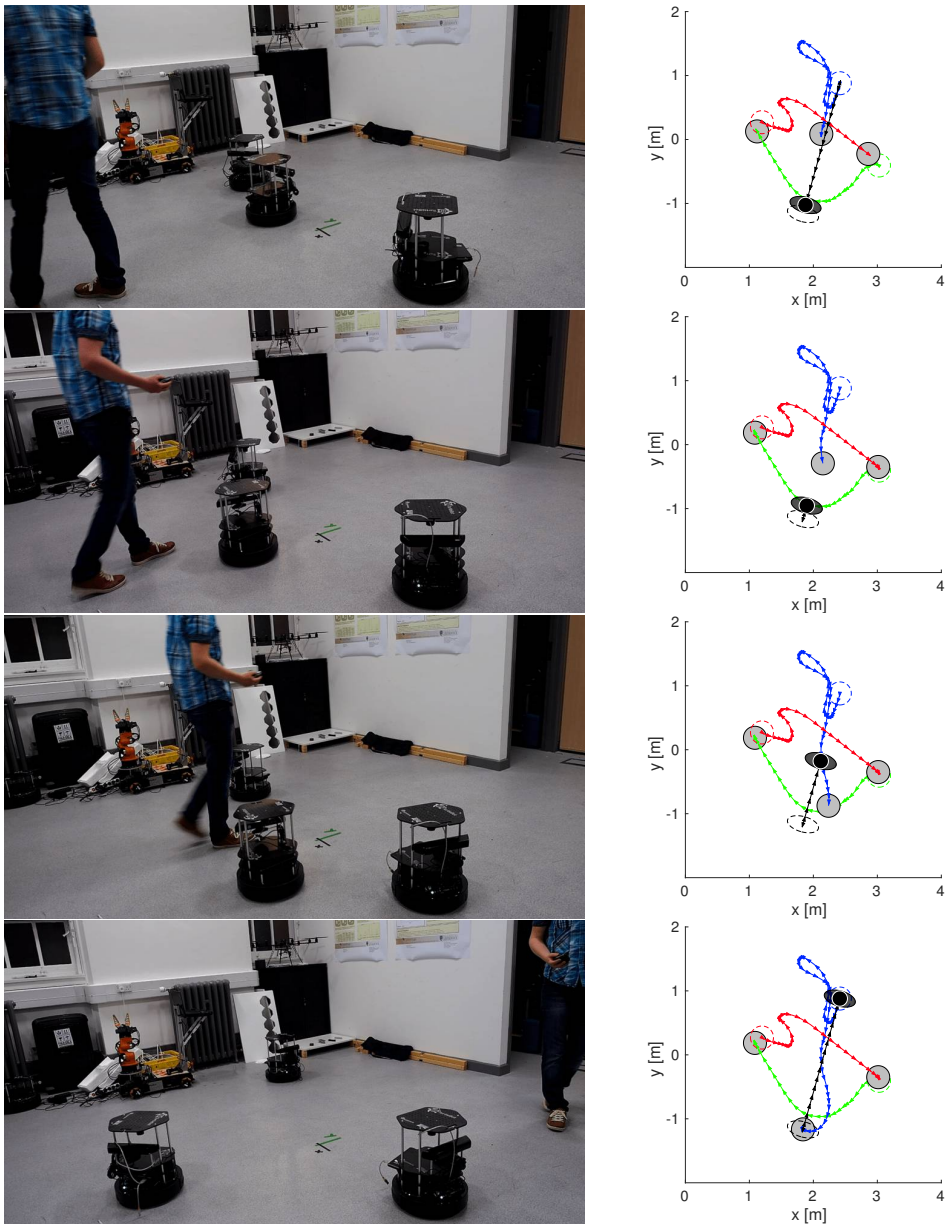


(a) (b)

FIGURE 3.18: Collision avoidance with three robots and a human. The human walks through the crowd of robots three times while the robots avoid him while driving to their target positions. (a) Pictures of the actual run. (b) Trajectories of the robots and the human. The human trajectory is approximated.

In our first experiment (Figure 3.17), we tested a robot exchanging the positions with a human. The pictures of the run are shown in the left column (Figure 3.17a) and the trajectories are shown in the right column (Figure 3.17b). The trajectory of the human is approximated. The human does not take care of avoiding the robot, he walks at a reasonable pace towards the robot. The robot detects the human and realises that it is on collision course. Thus, the robot avoids him by adjusting its path accordingly.

In a second experiment, we tested the antipodal circle experiment with one human and three robots. The human passes through the crowd of robots three times back



(c) (d)

FIGURE 3.18: Collision avoidance with three robots and a human (continued). The human walks through the crowd of robots three times while the robots avoid him and drive to their target positions. (c) Pictures of the actual run. (d) Trajectories of the robots and the human. The human trajectory is approximated.

and forth, while the robots have to reach the antipodal position (see Figure 3.18). The pictures of the run are shown in Figure 3.18a and Figure 3.18c and the trajectories are shown in Figure 3.18b and Figure 3.18d.

The robots that are not on a head-on collision course with the human, marked with red and green traces, start detecting the human and slow down. The robot that has to exchange the place with the human (blue traces) backs away in order to avoid the collision. The result is that the robot with red traces has to back away further, while the robot with green traces now has a free path towards his goal. After the human has

passed for the first time, the robot with red traces has a free path to its goal position, while the robot with blue traces has avoided the human by driving towards the to right corner. Afterwards, the human returns to his starting point, thus the robot with blue traces has a free path towards its goal. At the third pass, the robot with blue traces has to avoid the human once more. A video showing the results can be found at: http://wordpress.csc.liv.ac.uk/smartlab/collvoid/.

3.5 Summary

This chapter introduces a navigation approach that can deal with non-mapped static obstacles and dynamic obstacles such as humans and other robots. In our previous contribution, we introduced how the particle cloud of AMCL can be used as an estimator for the robots' localisation uncertainty, when the robots' footprints can be approximated by a circle. In the extension we allow for any kind of footprint by using the Minkowski sum of the convex hull of the actual footprint and a subset of the particle cloud of AMCL. The robots' footprint can then be enlarged accordingly to ensure safety in multi-robot situations.

Our new approach also introduces the use of a global planner to overcome deadlock situations and improve the performance in the presence of static obstacles. Additionally, we have presented how we can use Monte Carlo sampling to select the velocities in the velocity space according to some cost functions. This allows us to easily incorporate humans in the system and tune the cost functions, such that the robots keep more distance from humans than from other robots in order to not intrude the humans' personal spaces.

Lastly, we have shown that this sampling based method can be readily added to the well known and commonly used DWA planner in the ROS framework. This allows us to use the configuration space for avoiding static obstacles and integrating the VO-based avoidance in the velocity space for dynamic obstacles.

The presented methods have been extensively evaluated in simulation and in real world settings, illustrating the feasibility of the proposed approach.

Chapter 4

Spatial Task Allocation Problems

In the previous chapter, we developed a collision avoidance system than can be applied in Multi-Robot Systems (MRS). As mentioned in Chapter 1, decision making is another fundamental component for any MRS, as it allows the robots to act autonomously according to their current state.

In this work, we focus on problems in which a team of agents needs to service a set of tasks that are spatially distributed in an environment. Each task can be performed by one or more agents, and new tasks can appear in the world due to exogenous events outside of the agents' control.

For instance, consider the warehouse commissioning example introduced in Section 1.1. Another example is a cleaning task in, for instance, an office building. Dirt can appear at any location at any given time in this world, since a human can spill coffee, or leave paper on the floor, etc. A team of cleaning robots, in this case the agents, has the task to continuously keep the building clean.

In many cases, the decision making still has a centralised component, e.g. approaches using centralised planning or a single auctioneer for task allocation (Beetz et al., 2011; Zhang and Parker, 2012). These robots have limited autonomy and rely on the central control component of the system for planning purposes. Such approaches have two major limitations, namely robustness and scalability. If a centralised planning algorithm fails, for example, the whole system is affected and becomes unavailable. Second, for each additional robot to be added, the joint action and state spaces increase exponentially, rendering it unfeasible for any central algorithm to solve larger problems optimally in general.

Trying to overcome these issues by decentralised solutions introduces an important challenge in autonomous coordination of the multiple robots in the system. There are different approaches to tackle this problem, for instance swarm approaches inspired by nature (Lemmens and Tuyls, 2012; Alers et al., 2014; Brutschy et al., 2014), or others based on decentralised auctions (Capitán et al., 2013; Amador et al., 2014; Choi et al., 2009), in which each robot bids for tasks and task assignment ultimately follows from the winning bid values. The swarm approaches are generally reactive approaches that lack the possibility to plan ahead (e.g., when tasks may appear stochastically in

the environment) and the auctioning approaches rely heavily on reliable communication during the bidding phase. Furthermore, after the tasks are auctioned off, agent policies are fixed and remain unchanged until the next bidding phase.

In order to avoid these issues we develop a generic algorithm that applies to various instances of the coordination problem. We build on the idea from mean field game theory (Guéant et al., 2011) that for many tasks it may be sufficient to reason about the aggregate effect of the other agents. This idea has been proven to be useful in particular instances of off-line planning for large agent teams, such as congestion-like games and resource-coupled multi-agent planning (Gordon et al., 2012; Varakantham et al., 2012; Ahmed et al., 2012). In this chapter, we demonstrate that these ideas also bring great improvements in effectiveness in a very general class of online multi-robot planning problems. The key idea is to aggregate the effect of other robots in the area by using their locations as a proxy to predict their future actions and then choosing the best response. Using this principle, a robot does not need to reason about every other individual robot but only needs to know in general which tasks are likely serviced by another robot, in order to reduce the complexity of its own planning. Because planning is done online and at every time-step, the algorithm is flexible in responding to changes in the environment. This also implies that communication between robots is not strictly required, however, the robots do need to be able to observe the current state (i.e., locations of the other robots), which can be enabled through (local) communication.

We define the problem as a sub-class of Multi-Agent Markov Decision Processes (MMDPs) (Boutilier, 1996) that we collectively refer to as *Spatial Task Allocation Problems (SPATAPs)*, and develop our algorithm as a number of online planning approximations that are tailored to exploit the characteristics of these problems. In particular, we investigate the general algorithm for settings with *negative interactions* in which each task can be serviced by a single agent. In such tasks, it makes sense to discount the reward for tasks that are likely attended to by other agents. These considerations are related to the field of plan and intent recognition (Sukthankar et al., 2014).

Such approximate *empathetic* reasoning was recently exploited in the context of multi-robot exploration (Matignon et al., 2012) by making use of a modification of distributed value functions (J. G. Schneider et al., 1999) (henceforth referred to as *MDVF*). We introduce a simplification of this algorithm, *empathy by fixed weight discounting (EFWD)*, that applies to all SPATAPs with negative interactions. While these subjective approximations bring improvements in planning efficiency, they are not sufficient to make the planning problem tractable.

Another crucial contribution of our approach is therefore the combination of the algorithm with another approximation method, called *Phase-Approximation* (Claes et al., 2015), that forms the basis of an approximate online planning technique without any exponential dependence on the number of agents or the number of state factors.

Finally, an empirical evaluation shows that these combined techniques yield near-optimal solutions in cases in which the optimum can be calculated and is highly scalable,

while outperforming the state-of-the-art.

The remainder of this chapter is structured as follows. Section 4.1 summarises the related work; Section 4.2 recaps the theoretical background and the outlines the approach. Section 4.3 introduces the new SPATAPs framework and Section 4.4 presents the approximations that can be used to tackle these kinds of problems. Section 4.5 presents the empirical results of the approaches. Section 4.6 concludes the chapter and discusses future work.

4.1 Related Work

In this chapter, we define approximate models which we can solve optimally. This should be contrasted with efforts to approximately solve exact models, e.g. (Kocsis and Szepesvári, 2006). Combining such approaches (approximately solving the approximate models) may lead to even better scalability, required for real-life problems, which we will investigate in the following chapter.

The restrictions of the local problem of each agent to a subset of state factors is reminiscent of converting to a Dec-MDP (Bernstein et al., 2002), but in fact fundamentally different, since *the observation of the global state is used to construct the agents' subjective approximations.* Moreover, despite recent advances, e.g., (Dibangoye et al., 2012), Dec-MDP solution methods do not nearly scale to problems of the size considered here, or are suitable only for transition and observation independent settings (Becker et al., 2003; Dibangoye et al., 2014) (which our setting is not).

While there have been other approximate methods for solving MMDPs, these typically depend on pre-specifying the fixed, or context-dependent coordination structure (Guestrin et al., 2002; Kok, 2006; Spaan and Melo, 2008). For SPATAPs, however, fixed coordination structures are a poor choice and the number of contexts to be considered is huge. In addition, these methods are not aimed at exploiting the particular structure present in SPATAPs, independence of movement and locality of tasks. To overcome the problem of pre-specifying interaction structures one can try to learn them (Melo and Veloso, 2009; De Hauwere et al., 2010), but the premise underlying these methods is that there are only few states in which the agents need to coordinate. In contrast, in SPATAPs, the agents need to coordinate their task selection in *all* states.

Another perspective for decision making can be nature inspired swarm approaches. These are based on ant and honeybee colony behaviour and rely on local interactions in the environment to achieve coordination in MRS. From these local interactions global intelligence emerges at the group level, i.e. self-organisation, capable of achieving efficient coordination in foraging and coverage tasks. Although these swarm solutions are efficient and effective for foraging and coverage problems, they do not consider the dynamic appearance of new or different (sub)tasks that can arrive at unpredictable moments in the environment (Lemmens and Tuyls, 2012; Alers et al., 2014; Brutschy et al., 2014). Furthermore, (Brutschy et al., 2014) considers sequential interdependent tasks in which

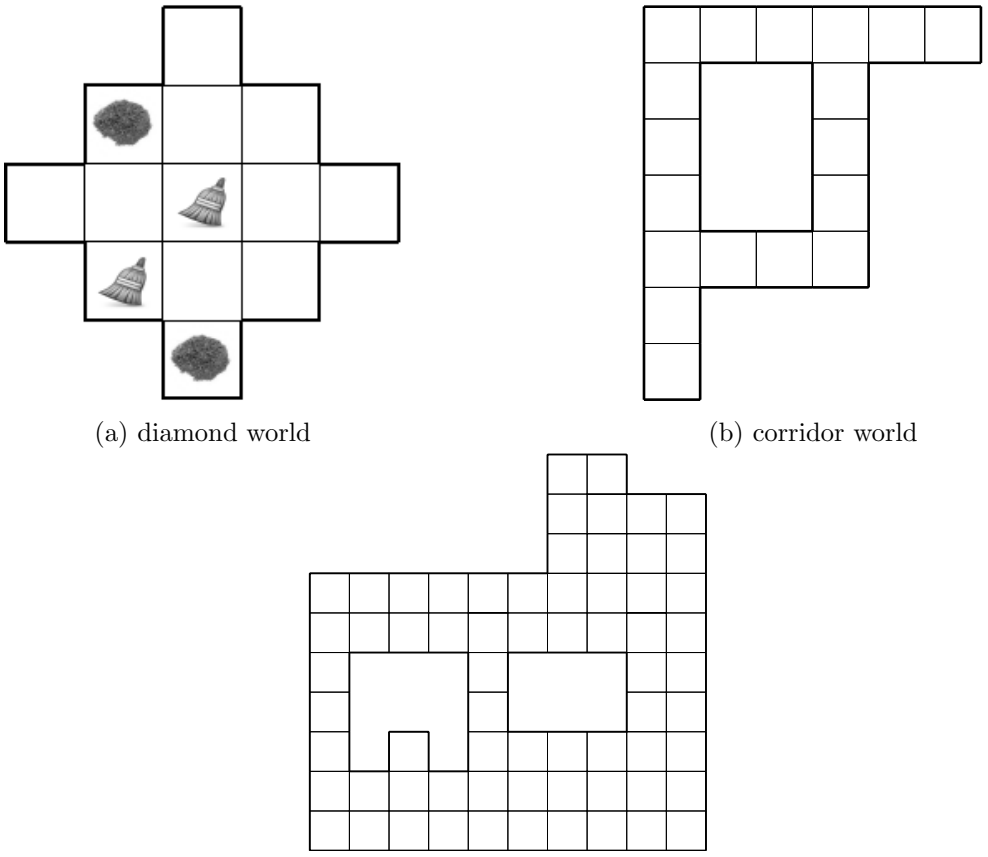
sub-tasks must be completed one after the other in order to complete an overall task. This work however is limited to the foraging problem and two sub-tasks.

Approaches for assigning agents to tasks based on auctions (Capitán et al., 2013; Amador et al., 2014) are closely related, but either do not reason about subsets of tasks (Capitán et al., 2013), or do not properly address the sequential nature of the task in SPATAPs (Amador et al., 2014; Choi et al., 2009). SPATAPs also relate to more general resource allocation problems (Wu and Durfee, 2010) (agents can be interpreted as resources). We, however, allow reallocation at every time-step and consider spatially distributed tasks and travel times.

Other approaches like SCRAM (MacAlpine et al., 2014) and the Hungarian method (Kuhn, 1955) are shown to work well for static assignment problems. In (Hanna et al., 2016) both methods are adapted to be applied in a car-sharing application in which the users are matched to available cars. This is closely related, but these do not consider the possibility of changing an assignment online during the execution. Once the assignment is made, this remains until the resource becomes available again. Additionally, these approaches are centralised, thus they are out of scope for our desired application.

The idea of interacting with the aggregate effect of other agents is studied in detail in the field of mean-field games (Guéant et al., 2011), where the focus lies on characterising equilibria. The idea is that when interacting with a large group of other agents, it is not necessary, but also not feasible, to simulate the complex behaviour of each individual agent. The effect of the other agents are progressively lost in the crowd, when looking from the planning agent's perspective. A few approaches have tried to extend these ideas to engineering settings such as taxi-fleet optimisation (Varakantham et al., 2012; Ahmed et al., 2012) and theme park crowd management (Gordon et al., 2012) via off-line planning. However, since these approaches perform off-line planning, these approaches are restricted in the richness of the state space that can be used as the basis for action selection. The 'aggregate effect' in these approaches typically consists of the number of agents present in different zones which directly affects utility for the protagonist. In our case, we use the predicted future agent locations as a proxy for their behaviours, which in turn will affect the utility of the protagonist.

Finally, the subjective approximations presented in this chapter can be interpreted as online planning for a special instance of a level 1 interactive Partially Observable Markov Decision Process (POMDP) (Gmytrasiewicz and P. Doshi, 2005; F. Doshi and Roy, 2008). In contrast to standard interactive POMDP solution methods, however, we propose dedicated approximation algorithms that exploit the characteristics of SPATAPs by using location as the proxy for the other agents' policies.



(a) diamond world

(b) corridor world

(c) office world

FIGURE 4.1: Three possible SPATAP environments.

4.2 Spatial Task Allocation Problems

We consider a problem class that we refer to as SPATAPs. In particular, SPATAPs describe settings in which a team of agents needs to service a set of tasks that are spatially distributed in an environment. Each task can be performed by one or more agents, and new tasks can appear in the world due to exogenous events outside of the agents' control.

As a running example we will consider the cleaning task as described in the introduction. A team of robots has to clean an office building. There is no central control of any sort, but the robots can observe the location of other robots, e.g. by using overhead cameras or communication. We will assume that the office building can be represented using a grid world, where each tile corresponds to a part of a room of a fixed size, but other representations (such as a more abstract graph) are also possible. The grid worlds can have arbitrary shape, see Figure 4.1 for examples that we consider in this paper.

The main problem is how the agents should plan their behaviour in order to ensure that all tasks are serviced, i.e., in this case, that all dirt is continuously cleaned while minimising interference between robots.

Since there are nearly no restrictions on the sort of tasks, SPATAPs provide a very powerful and general model which are very hard to solve optimally. This is due to the

fact that the joint action and state spaces increase exponentially with the number of agents in the system.

In fact, only a few methods can deal with such large problems. One approach that offers the required scalability is 'partition organisation' (Sleight and Durfee, 2012). In this approach, the problem is partitioned in (overlapping) regions and each agent is assigned to one region. This approach is well suited for problems in which there is a straightforward partitioning, such as symmetric worlds. However this is not always possible, especially when the task appearance probabilities are not evenly distributed over the world or when specific tasks can only be served by particular agents.

Another common approach, which we will treat in some more detail in Section 4.4.1 under the name *self-absorbed approximation*, is to ignore the presence of other agents during planning and to consider them as mere noise to each individual planning model. In the case of task allocation problems, however, such approximations lead to poor performance (as supported by our empirical evaluation, see Section 4.5), since the difficulty in this type of problems revolves around the coordination of which agent addresses which task.

We propose to exploit the key characteristics of SPATAPs—independence of agent movement and the locality of tasks—for *efficient approximations* during online planning. The key idea is that an agent does not need to reason about all other agents individually, but can reason about their aggregated effect. In particular, if another agent is close to a task location, we can reason that this task will be serviced with a certain probability at some future time-step. If we assume a reasonable policy for the other agents we can predict the probabilistic movements for all agents over the planning horizon and use the aggregated probability distributions of the locations of the other agents as a sufficient statistic for calculating our best response.

Algorithm 6 summarises the idea described above. For all other agents, we fix a policy π and, for the time span of our planning horizon, predict their movements as following π. By doing this we obtain a probability distribution of the agents' location for all time-steps, i.e., we compute p_j^t, which is the probability distribution of the location of agent j at time-step t. These probability distributions can be aggregated and used by each individual agent to calculate a best response, which we refer to as *presence mass (pm)*.

In the next section, we show that SPATAPs can be directly modeled as MMDPs, and explain why this is not of immediate help since the complexity of solving the problems with standard MMDP methods remains prohibitively high. Section 4.4 presents a remedy by introducing online approximate methods that exploit the key characteristics of SPATAPs.

Algorithm 6: Aggregated best response

Input :
i : planning agent i
$\mathcal{D}_{-i}$: set of other Agents
h : planning horizon (time-steps)

foreach $t \in h$ **do**
  foreach $j \in \mathcal{D}_{-i}$ **do**
    compute estimated policy π

foreach $t \in h$ **do**
  //aggregate
  $pm = \sum_{j \neq i} p_j^t$

compute best response of agent i given pm

4.3 Spatial Task Allocation Problems Formulated as Multi-Agent Markov Decision Processes

The problems we consider in this chapter describe a set of spatially distributed tasks that a team of agents or robots needs to solve. A key characteristic of such problems is that the outcome of an action is uncertain (cleaning the dirt or moving in the world may each fail with some probability, e.g., due to wheel slip), and new tasks can appear due to unforeseen exogenous events (e.g., a human spilling some dirt). Therefore, SPATAPs can be seen as a special case of MMDPs, as we will show in the following.

As introduced in Chapter 2, Markov Decision Processes (MDPs) provide a general framework for sequential decision making under uncertainty (Puterman, 1994). Revisiting Definition 2.2, the MMDP is the straight-forward extension to the case of multiple decision makers who observe the full state of the environment.

In this chapter, we will consider (undiscounted) h-stage look-ahead planning, i.e., constructing a plan that specifies actions from 'now', $t = 0$, to stage $t = h - 1$. For this setting, the value function for each stage t, when following policy π equals $V_t^\pi(s) = \max_a Q_t(s, a)$, where

$$Q_t(s, a) = R(s, a) + \sum_{s'} P(s'|s, a) V_{t+1}(s'). \tag{4.1}$$

The optimal policy π^* and corresponding optimal value functions Q_t, maximize the expected reward for every (s, a).

SPATAPs can be defined as a sub-class of MMDPs with some additional structure. Underlying a SPATAP is a map that specifies the potential task locations $\mathcal{L}$ and that defines $\mathcal{A}_M$, the set of movement actions. E.g., for the "dirt cleaning" example, all agents are homogeneous and share a common (movement) action space $\mathcal{A}_M = \{N, E, S, W, STAY\}$. There further exists a *task structure*, defined by a set of task types $\mathcal{T} = \{\mathcal{T}_0, \mathcal{T}_1, \ldots, \mathcal{T}_{|\mathcal{T}|}\}$. Each type $\mathcal{T}_k$ has an associated set of task states $\mathfrak{T}_k$ that indicate the status of the task. In our running example, $\mathcal{T}_1$ could have states

$\mathfrak{T}_1 = \{\text{very dirty, dirty, nearly clean}\}$. $\mathcal{T}_0$ refers to a special type indicating there is no task and only has one state $\mathfrak{T}_0 = \{CLEAR\}$. We use $\mathfrak{T} = \bigcup_k \mathfrak{T}_k$ to denote the set of all task states. Each task type $\mathcal{T}_k$ may optionally be associated with one (or more) particular action $a_{\mathcal{T}_k}$ to perform that task. Each agent i can perform movement and task actions. The agent team can be homogeneous, i.e., have the same capabilities, but the framework also allows agents that have different capabilities, or differences in how effective an agent is at a particular task. The approximation method we introduce in Section 4.4.1, does assume homogeneous agents, but the approach can be extended to heterogeneous teams by applying the same technique to each 'type' of agent.

These SPATAP-specific components can now be used to define the induced MMDP. Agents and their actions are unchanged. A state is a tuple $s = \langle \lambda, \tau \rangle$, where λ is the vector of locations (λ_i denotes the location of agent i), and τ is the task status vector (τ_x denotes the task status at location x). The transition function can be factored as

$$P(\lambda', \tau' | \lambda, \tau, a) = \prod_{x \in \mathcal{L}} p_x^T(\tau_x' | \tau_x, \lambda, a) \prod_{i \in \mathcal{D}} p_i^M(\lambda_i' | \lambda_i, a_i) \tag{4.2}$$

where p^T are task transition probabilities and p^M are agent movement probabilities. Movements are independent, effectively assuming that lower-level path-planning will avoid collisions within the same location.

The task transition probabilities p^T are assumed to be conditionally independent given the locations and actions of the agents and encode the probability of progressing toward finishing the tasks, as well as exogenous events that spawn new tasks.

The reward function is additively factored and is the sum of task rewards R^T and movement costs R^M:

$$R(s, a) = \sum_{x \in \mathcal{L}} R_x^T(\tau_x, \lambda, a) + \sum_{i \in \mathcal{D}} R_i^M(\lambda_i, a_i). \tag{4.3}$$

As the name 'movement cost' implies, the functions R_i^M will typically specify negative rewards. For the task reward functions, it is also natural to specify a cost for every stage that the task is not completed (consider the robot rescue setting where victims require attention as quickly as possible). More options are possible, e.g., a business may want to use as the reward function the actual amount of money a particular assignment will generate, or in the warehouse commissioning domain, the time an order has been waiting. Note that the model also supports rewards that depend on the next state by taking the expectation over next state, as can also be seen in Equation 2.5. E.g.,

$$R_x^T(\tau_x, \lambda, a) = \sum_{\tau_x'} P(\tau_x' | \tau_x, \lambda, a) R_x^T(\tau_x, \lambda, a, \tau_x'). \tag{4.4}$$

Locality Assumptions

While the above lays out the general form of SPATAPs, such problems are, in general still very difficult since the terms p_x^T and R_x^T are *non-local*, i.e., they depend on all the agents, including those that are far away from location x. In order to gain more traction on the problem, we will assume that a task at a particular location x will only be influenced by a subset of agents. This subset we call the *locality scope* $\mathbb{L}(x, \tau_x, \lambda, a)$ and it depends on the location x, the task type and state encoded by τ_x, and λ, a. For instance, in our example $\mathbb{L}(x, \tau_x, \lambda, a)$ for a dirty location x will only contain those agents at x that perform the 'clean' action. In the remainder of this chapter, we will simply write $a_{\mathbb{L}}$ for the action profile of agents in the locality scope. As such, we will consider task transitions of the form $p_x^T(\tau_x'|\tau_x, \lambda_{\mathbb{L}}, a_{\mathbb{L}})$. Similarly, we will assume that the task rewards can be expressed as $R_x^T(\tau_x, \lambda_{\mathbb{L}}, a_{\mathbb{L}})$.In this chapter, we use a $R_x^T(\tau_x, \lambda_{\mathbb{L}}, a_{\mathbb{L}})$ formulation of the rewards, because that permits taking the expectation of next-stage task states, analogous to (4.4).

Each SPATAP is an MMDP, so MMDP solution methods apply. Unfortunately, both the number of states and joint actions are very large for these problems (see Table 4.1). As a result, even state-of-the-art MDP solvers that exploit factored structure (Hoey et al., 1999) have problems with the smallest SPATAPs. One would hope that the special structure of SPATAPs may make them easier to solve but, unfortunately, this is not the case in general:

Theorem 4.1. *Optimally solving SPATAPs is as hard as solving MMDPs; i.e., SPATAPs are 'MMDP-hard'.*

Sketch 1. We reduce from the problem of solving an MMDPs by creating a SPATAP with a single location and a single task. The task states correspond to the states of the MMDPs and similarly can we derive p_x^T and R_x^T from the transitions and reward of the MMDP. The optimal solution of this SPATAP is the optimal solution of the MMDP.

This theorem illustrates that while the concept of tasks is very general and powerful, this comes at a worst-case computational cost. Nevertheless, SPATAPs offer ample opportunities to exploit their specific characteristics. In particular, in the following sections, we propose two types of approximation techniques that each directly exploit problem structure.

4.4 Online Approximations for Solving Spatial Task Allocation Problems

We introduce two orthogonal approximation approaches for online planning for SPATAPs, that increase efficiency by exploiting two characteristics of real-world SPATAPs.

First, SPATAPs exhibit independence of movement and locality of task transitions and rewards, which implies that agents in many cases can act relatively independently of each other. To exploit this property, we consider subjective approximations that reduce

the complexity introduced by the large number of agents. In particular, we propose a novel *empathetic* subjective approximation method that combines the computational benefits of subjective approximations with an approximate reasoning over the team members as introduced in Algorithm 6.

Second, for SPATAPs that exhibit only infrequent appearance of new tasks, planning about the currently active tasks can be expected to yield good performance. We propose to exploit this using a novel 'phase approximation' technique, which reduces the complexity introduced by the large number of states.

Both techniques transform larger input MMDPs into approximate, smaller output (M)MDP models that leverage the unique properties of SPATAPs. The resulting models can be solved optimally using standard online planning methods. Moreover, both techniques are highly complementary and their combination yields approximate solution methods that are robust, scalable, and easy to implement: each agent can simply use its individual, online (single-agent) MDP planning method applied to a *subjective phase MDP*, which leverages the techniques described above.

In this chapter, we assume that the agents use the resulting MDPs to perform online planning over a fixed look-ahead horizon (h), but more sophisticated (e.g., adaptive horizon (Droge and Egerstedt, 2011)) methods can be applied too.

4.4.1 Subjective Approximations

The first set of techniques by which we bring computational leverage to the (online) planning process is the implementation of the approach described in Section 4.2, which aims to address the complexity due to the presence of multiple agents. They increase planning efficiency by distributed approximation: decomposing the larger problem into a set of approximate smaller planning problems, one for each agent.

Self-absorbed Agent Approximation

The extreme case of subjective approximation is to plan for each agent independently, assuming that it is the only agent present in the problem. This is a common approach (Gmytrasiewicz and P. Doshi, 2005), and we refer to this type of approach as the *'self-absorbed agent' approximation*. A self-absorbed agent i only models its own location and thus has individual states $s_i = \langle \lambda_i, \tau \rangle$. It also assumes that the transitions only depend on its own actions.

Formally, we define a *subjective MDP (S-MDP)* for agent i as a tuple $\langle \mathcal{S}_s, \mathcal{A}_i, p_i^{SA}, R_i^{SA} \rangle$, where $\mathcal{S}_s$ is the subjective state space of states $s_i = \langle \lambda_i, \tau \rangle$, $\mathcal{A}_i$ is the space of individual actions for agent i, p_i^{SA} and R_i^{SA} as the self-absorbed transition reward functions as follows:

$$p_i^{SA}(s_i'|s_i, a_i) = \left[\prod_{x \in \mathcal{L}} p_x^{T,SA}(\tau_x'|\tau_x, \lambda_i, a_i) \right] p_i^M(\lambda_i'|\lambda_i, a_i)$$

TABLE 4.1: Sizes of the state and actions spaces of the considered models. $|\mathcal{D}|$ is the number of agents, $\mathcal{A}_*$ denotes the largest individual action set. Planning times are a polynomial function of these quantities.

	state space	action space
MMDP	$\lvert\mathcal{L}\rvert^{\lvert\mathcal{D}\rvert} \cdot \lvert\mathfrak{T}\rvert^{\lvert\mathcal{L}\rvert}$	$\lvert\mathcal{A}_*\rvert^{\lvert\mathcal{D}\rvert}$
S-MDP	$\lvert\mathcal{L}\rvert \cdot \lvert\mathfrak{T}\rvert^{\lvert\mathcal{L}\rvert}$	$\lvert\mathcal{A}_*\rvert$
Phase-MMDP	$\lvert\mathcal{L}\rvert^{\lvert\mathcal{D}\rvert} \cdot \lvert\mathfrak{T}\rvert^{\lvert p\mathcal{L}\rvert}$	$\lvert\mathcal{A}_*\rvert^{\lvert\mathcal{D}\rvert}$
SP-MDP	$\lvert\mathcal{L}\rvert \cdot \lvert\mathfrak{T}\rvert^{\lvert p\mathcal{L}\rvert}$	$\lvert\mathcal{A}_*\rvert$
k-SP-MDP	$\lvert\mathcal{L}\rvert \cdot \lvert\mathfrak{T}\rvert^{k}$	$\lvert\mathcal{A}_*\rvert$

$$R_i^{SA}(s_i, a_i) = \left[\sum_{x \in \mathcal{L}} R_x^{T,SA}(\tau_x, \lambda_i, a_i)\right] + R_i^M(\lambda_i, a_i). \tag{4.5}$$

It may be difficult to map $p_x^T(\tau_x'|\tau_x, \lambda_{\mathbb{L}}, a_{\mathbb{L}})$, $R_x^T(\tau_x, \lambda_{\mathbb{L}}, a_{\mathbb{L}})$ to $p_x^{T,SA}(\tau_x'|\tau_x, \lambda_i, a_i)$ and $R_x^{T,SA}(\tau_x, \lambda_i, a_i)$ respectively. However, in many cases, it is possible to assume a default effect or default action for the other agent (e.g., we can assume that there will be no other agent cleaning the same spot). Another approach is to treat the agents as noise (Gmytrasiewicz and P. Doshi, 2005), by imposing some (e.g., uniform) distribution over λ_{-i}, a_{-i}.

Solving an S-MDP can be done with standard techniques, yielding value functions $V_{i,t}^{SA}(s_i)$ and $Q_{i,t}^{SA}(s_i, a_i)$, which directly follow from (4.1).

The S-MDP improves significantly over the MMDP formulation in terms of complexity (see Table 4.1). As shown, there is no longer any exponential dependence on the number of agents in an S-MDP, which directly means that it admits more efficient solutions. However, we expect that self-absorbed agent approximations are insufficient in domains where agents need to perform a fair amount of coordination. Next, we propose a number of approaches that do account for interactions between agents.

Empathy by Predicting other Agents' Locations

As summarized in Algorithm 6, the key idea is the following: in order to compute a best-response from the perspective of one agent, it only needs to predict what tasks will be tackled by the other agents. That is, it only cares about the aggregate effect of the actions of the rest of the team, but not about which team member addresses which task in particular. In order to predict what tasks will be addressed by the rest of the team, we use sum of the predicted probability of the other agents being at a location as a proxy for them addressing the task at that location. We refer to this as *presence mass (pm)*.

In particular, from the perspective of an agent i, we want to be able to predict the location λ_j^t of all other agents $j \neq i$, t-stages from now. That is, we want to compute the probability distribution $P(\lambda_j^t|s^0)$ where s^0 is the full MMDP state 'now' (i.e., at the time when the agent performs this prediction).

To compute these 'presence mass' distributions, one needs to assume particular behavior of the other agents. One possibility is to assume that other agents perform a random walk (Matignon et al., 2012). This assumption, however, leads to uninformative uniform distributions over states when predicting further into the future. To avoid this problem, we assume that the other agents use a self-absorbed model with quantile response (i.e., we assume that the other agents take actions according to a Boltzmann distribution (Sutton and Barto, 1998) specified using the self-absorbed agent approximation V^{SA}):

$$P_{i,t}(a_i, s_i) = \frac{e^{\frac{Q^{SA}_{i,t}(s_i,a_i)}{T}}}{\sum_{b \in \mathcal{A}_)} e^{\frac{Q^{SA}_{i,t}(s_i,b)}{T}}}, \tag{4.6}$$

where T is a constant temperature and $P_{i,t}(a_i, s_i)$ defines the probability of an action for the other agents, based on their self-absorbed state, s_i. Figure 4.2b illustrates that this leads to more sensible predictions. When there are multiple agents present, we can accumulate the presence mass distributions into a single 'sufficient statistic' and therefore do not need to account for every single agent during planning. We use the accumulated presence mass distribution for the next model.

Empathy by Fixed Weight Discounting

After calculating the aggregated presence mass of the other agents, the planning agent has to choose its response. Distributed value functions (DVFs) (J. G. Schneider et al., 1999) allow agents to share their value function. This, however, leads to a lot of communication overhead. The MDVF (Matignon et al., 2012) approach is inspired by DVFs and implements agent collaboration by using a second value function (specifically, V^{SA}) to discount the values of future states. In the resulting formulation, however, MDVF agents do *not* share their value functions. Instead, each agent computes V^{SA} in parallel and uses it to discount the V^{MDVF} values.

Realizing this, we propose a more straightforward approach: we do not discount using V^{SA}, but just use the next-stage value function. We refer to this simplification as *empathy by fixed-weight discounting (EFWD)*. The resulting value function is given by

$$Q^{FWD}_{i,t}(s_i, a_i) = R^{SA}(s_i, a_i) + \sum_{s'} p^{SA}(s'_i | s_i, a_i) \left[\left(1 - f_i \sum_{i \neq j} P(s^{t+1}_j = s'_i | s^0) \right) V^{FWD}_{i,t+1}(s'_i) \right]. \tag{4.7}$$

where R^{SA} and p^{SA} are the self-absorbed model components (these are the same for all agents and hence we drop the subscript i to simplify notation). The last probability term is the presence mass of an agent j being at the location specified by s'_i $t+1$ stages into the future, i.e., $\Pr(s^{t+1}_j = s'_i | s^0) = \Pr(\lambda^{t+1}_j | s^0)$ with λ^{t+1}_j the location specified by s'_i. Finally, f_i is a fixed weight that determines how much the value of a next state is

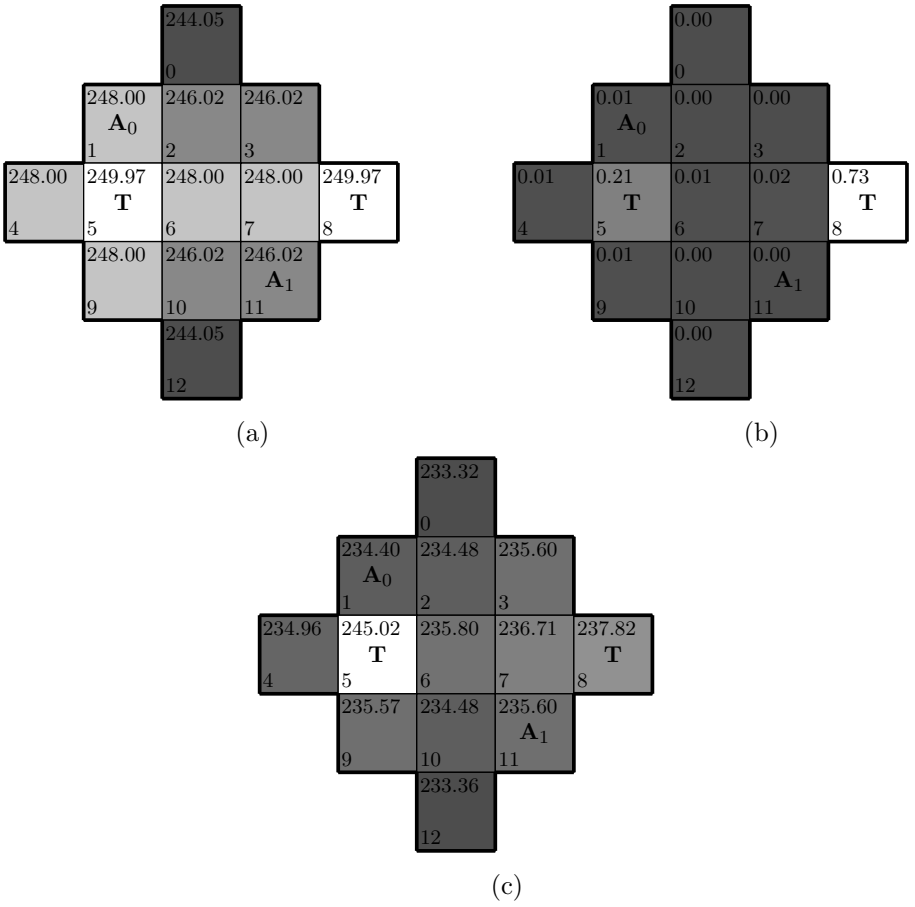


FIGURE 4.2: (a) A sample state for a diamond shaped gridworld with two agents (A_0, A_1) and two active task locations and V^{SA} for the current state. (b) The presence mass of A_1 from the viewpoint of A_0. (c) The discounted value function V^{EFWD} for A_0 resulting for this configuration.

discounted. We follow (Matignon et al., 2012) and set f_i to $\max R(s,a)/\max V(s,a)$. An example illustrating this discounting is shown in Figure 4.2.

4.4.2 Phase-Approximations

While subjective approximations reduce the complexity due to multiple agents, they do not sufficiently reduce the complexity of the state space. To overcome the complexity of the state space, we propose a different way of approaching the problem. Rather than seeing each location $x \in \mathcal{L}$ as a potential location for a task that may appear or disappear over the planning horizon, we focus only on the current 'task phase', i.e., the *currently active* set of tasks indicated by those locations x for which $\tau_x \neq CLEAR$. By focusing only on these locations, the number of task states induced is much smaller, allowing for big increases in planning efficiency.

Phase MMDPs

We formalize this idea by means of the so-called *phase-MMDP*, which, given a global state $s = \langle \lambda, \tau \rangle$, can be defined as follows. A *Phase-MMDP for state* s, is an MMDP $\langle \mathcal{D}, \mathcal{S}_p, \mathcal{A}, P^p, R^p \rangle$. The considered task locations in this MMDP, however, are restricted to the set of $p\mathcal{L} = \{x \in \mathcal{L} \mid \tau_x \neq CLEAR\}$ of *phase task locations*, i.e., the set of 'active' locations where there is a task. Thus, the state space $\mathcal{S}_p$ is spanned by the set $\mathcal{L}^{|\mathcal{D}|}$ of joint locations and the set $\mathfrak{T}^{|p\mathcal{L}|}$ of all possible task vectors for the active locations. We write $p\tau \in \mathfrak{T}^{|p\mathcal{L}|}$ for the restriction of τ to the locations in $p\mathcal{L}$. A phase-MMDP state is a tuple $ps = \langle \lambda, p\tau \rangle$. The transition (and reward) function follow from equation 4.2 (and 4.3) by restricting the product (summation) to $p\mathcal{L}$.

A phase-MMDP provides leverage by restricting the number of states compared to the regular MMDP formulation. However, this is highly dependent on the number of active tasks, and, in the worst case, it does not reduce the state space at all. Also, it does not address the large joint action space (see Table 4.1). This motivates the combination of the previously introduced techniques.

Subjective Phase MDPs

Realizing that subjective and phase-approximations yield complementary gains, we propose to combine both approximations in a formalism that we refer to as *subjective phase MDP (SP-MDP)*. An SP-MDP for agent i is a subjective model, meaning that it includes only the actions of agent i itself, moreover, it is a phase approximation, meaning that the states only include task states for active tasks. Specifically, a local state is a tuple $s_i = \langle \lambda_i, p\tau \rangle$, where λ_i is the location of agent i and $p\tau$ is the phase task vector. In an SP-MDP, the number of actions is the number of individual actions and the number of states is potentially much smaller due to the phase-approximations assumption (see Table 4.1).

kSP-MDPs

As mentioned, in the worst case there are many active tasks, which means that the number of states will still be prohibitive. However, by the combination of subjective and phase-approximations, it is possible to exploit the problem structure even further. In particular, *the subjective model of each agent may make different approximations* by exploiting what parts of the current state are relevant to that agent.

For instance, in the construction of the SP-MDP for an agent i, we can now make use of the location of that agent, by restricting the state space of the SP-MDP to include only task locations for the k nearest tasks. We refer to the resulting model as kSP-MDP. The number of states of the kSP-MDP is given by $|\mathcal{S}_{ksp}| = |\mathcal{L}| \cdot |\mathfrak{T}|^k$.

world	SA	MDVF	EFWD	SPUDD
2x2	93.32%	97.86%	98.41%	100%
3x3	94.73%	96.83%	97.24%	100%

(a)

world	$\|\mathcal{L}\|$	n	$\|\mathcal{S}\|$	$\|\mathcal{A}\|$
Line	12	2	$5.90e+05$	25
Diamond	13	3	$1.80e+07$	125
Corridors	18	3	$1.53e+09$	125
4x4	16	4	$4.29e+09$	625
6x6	36	5	$4.16e+18$	3125
Office	66	6	$6.10e+30$	15625

(b)

TABLE 4.2: (a) Relative values of the three approaches averaged across a set of randomly drawn starting states and compared to the SPUDD optimum value function. (b) Larger dirt-world benchmarks.

As is clear from Table 4.1, the kSP-MDP is *the only model* that is guaranteed not to have any exponential complexities. Standard dynamic programming for a h-step look-ahead MDP takes time $O(h|\mathcal{S}|^2|\mathcal{A}|)$, and thus is feasible for large problems when using the kSP-MDP model.

4.4.3 Social Laws to Overcome Symmetries

One of the drawbacks of subjective approximations is that when deploying a team of identical agents, this can lead to agents behaving identically when this is not desired. For instance, when two agents are in the same location, they will make the same assumptions about the other agents' behavior and thus compute the same value function. Consequently they will take the same action and (with probability depending on the movement model) end up in the same next location.

Although this phenomenon exposes a principal flaw of subjective approximations, in SPATAPs these issues are easy to deal with by employing social laws (Shoham and Tennenholtz, 1995). For instance, in our experimental evaluation we employ the following social law: whenever more than one agent is in the same location, these agents select their actions based on their IDs: the agent with the lowest ID selects the action with the highest expected reward, the next agent selects the second best action, etc.

4.5 Experiments and Results

In this chapter, we have introduced a number of approximations that culminated in a model without any exponential dependence on the number of agents or state factors. This model can therefore be efficiently solved online using standard MDP techniques. Since the approximations that we introduced are not bounded, we report the results

of an empirical evaluation aimed at determining the solution quality afforded by these approximations. For this purpose, we implemented a dirt-world simulation in which agents plan online, in a distributed fashion, using the kSP-MDP model.

The movement transition probabilities p_i^M are such that a movement can fail (the agent remains at its previous location) with 10% probability. The task at location x is deterministically completed if any agent i performs action $STAY$ at that location. A task appears at a location x with probability 0.05 (but an agent staying at a location prevents task appearance). The team of agents receive reward +1 for every clean location at every time step. We do not consider movement costs. Agents solve their individual kSP-MDPs for (a maximum of) $h = 20$ steps look-ahead, using regular dynamic programming using the MDVF, EFWD or SA methods.[1] Unless reported differently, we use the $k = 4$ nearest tasks.

In order to assess overall solution quality, we compare the approach with the global MMDP solution. Note that the global MMDP, unlike the phase-MMDP approximation, considers all locations on the board potential task locations, even currently 'inactive' ones. We use SPUDD (Hoey et al., 1999), the state-of-the-art optimal solver for factored MDPs, to provide the value of the optimal solution for horizon 10, and compare this to the average value generated by 100 dirt-world simulations with online planning[2]. Table 4.2a shows the results for this comparison. SPUDD was only able to scale to 2x2 and 3x3 gridworlds with two and three agents respectively. For these problems, the approximations perform very well; even the naive self-absorbed approximation achieves over 93 % of optimal. The proposed simplification EFWD even yields slightly higher rewards than the more complex MDVF.

In order to provide insight into the differences in behavior between the approaches, we investigate the rewards received over the different stages per episode. Figure 4.3 shows the mean reward per time-step (mean is taken over 10 episodes) in a 4x4 gridworld filled with tasks. As the figure shows, the reward increases at first and then converges close to the maximum reward achievable per time-step (which is 16 and received when all squares are clean). Interesting to note is that V^{SA} outperforms the empathetic approximations in the first time-steps but then converges to a lower average reward. This is probably again due to the lack of considering the effects of other agents when using V^{SA}. The agents first select the tasks greedily, thus close tasks, and the tasks further away are not covered. The empathetic approaches avoid this pitfall: by discounting the values of locations quickly reachable by many other agents, the locations only reachable by a protagonist are incentivised.

To examine the impact of restricting planning to only the k nearest tasks, we performed an experiment in which we vary k, holding other parameters fixed. For this experiment, we used a "full" 4x4 gridworld, i.e. dirt is present everywhere, with three agents

[1] Increasing the look-ahead beyond 20, did not increase the performance in our experiments. Shortening it does hurt performance in the larger problems, since it may lead agents to conclude that a task can never be addressed.

[2] Even though SPUDD is not specialized for MMDPs, it currently still is the best optimal solver for such problems. (Scharpff et al., 2013; Scharpff et al., 2016)

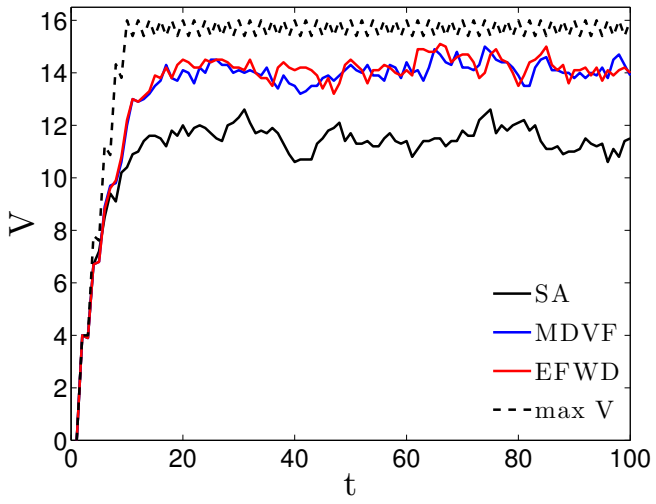


FIGURE 4.3: Mean reward over time for a 4x4 gridworld (initially full) with three agents.

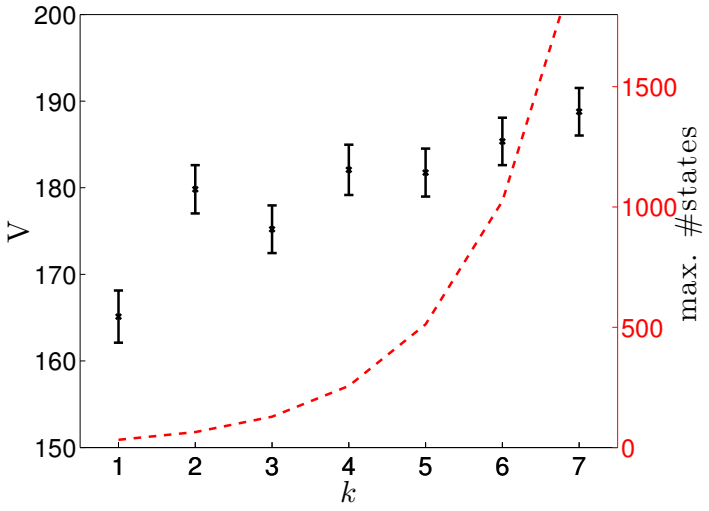


FIGURE 4.4: Mean reward and number states for k-nearest task phase MDP with a 4x4 gridworld and three agents while increasing k.

that used the E-FWD algorithm to select their actions. Results shown in Figure 4.4, are averaged over 100 runs of horizon 20 with 95% confidence intervals. Additionally shown are the number of states for each k (the dashed line). The figure clearly shows that, although $k = 1, 3$ perform poorly, there is no significant difference for $k \geq 4$, which explains our choice of $k = 4$ for all the other experiments. Finally, we test the performance of our approximations on a number of larger test problems, listed in Table 4.2b. The "Line" world is a straight line of 12 states, and the "Corridors" and "Office" world are shown in Figure 4.1. These problems are too large to be solved optimally, e.g., our largest test domain "Office" has $6.10e+30$ states and 15625 actions, which makes it well beyond anything that can be solved optimally.

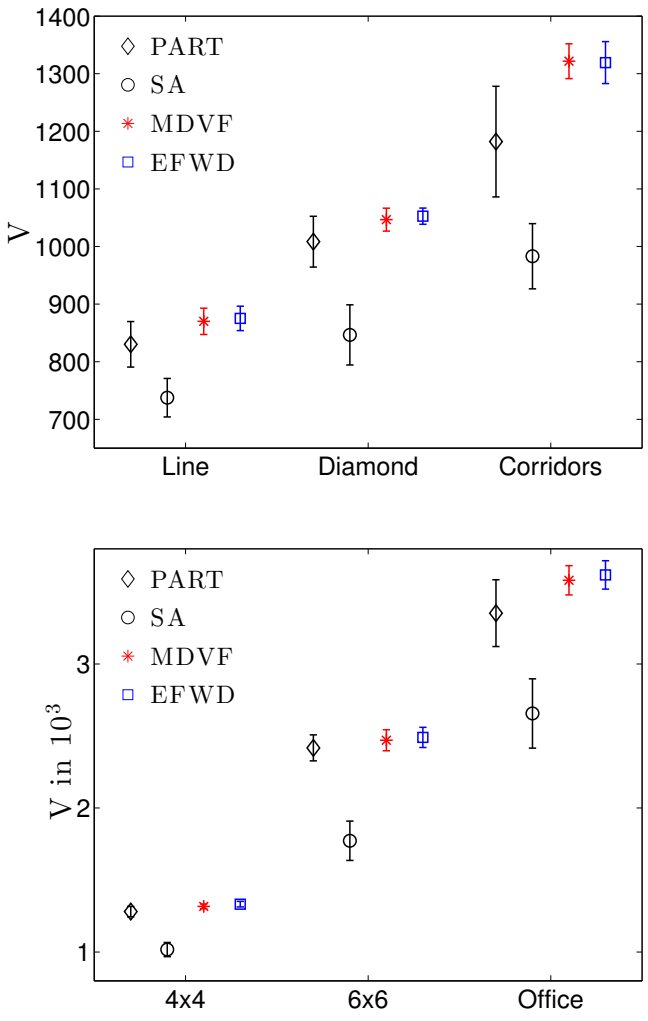


FIGURE 4.5: Mean reward for various gridworlds as presented in Table 4.2b .

We compare our approach against the "partition" approach as described in Section 4.2. We automatically calculate the partitions by assigning each location to the closest agent. If there are multiple agents with the same distances, the location is added to both partitions. We refer to this approach as "PART". PART still suffers from the fact that large regions lead to too large local problems which we addressed by also restricting to the k nearest tasks in these problems.

Each method is run for 100 steps and repeated 10 times with random initial positions for the agents, while the world always being "full". Figure 4.5 shows the mean total reward including the 95% confidence intervals, i.e. non-overlapping error-bars mean statistically significant results. The self-absorbed approach performs the worst for every setting. The simplifications of EFWD do not lead to a loss: there is no significant difference with MDVF and both have higher means and smaller variance than PART. Especially in more complex worlds, i.e. the "Corridors" and the "Office", the PART approach has a very high variance due to the different partitioning for each run. EFWD is more reliable because it does not depend on the initial partitioning.

world	U.B.	EFWD	% of U.B.
Line	1104.8	875.5	79.2%
Diamond	1227.9	1052.6	85.7%
Corridors	1657.2	1379.3	79.6%
4x4	1518.0	1332.1	87.8%
6x6	3242.9	2490.2	76.8%
Office	5137.7	3618.6	70.4%

TABLE 4.3: Performance of EFWD as compared to an upper bound for problems that cannot be solved optimally.

Additionally, we computed a theoretical (loose) upper bound by assuming that at every stage the expected number of tasks appears (fractions of tasks are allowed) and that all agents are able to clean an assigned task within two time-steps. In other words, the upper bound assumes that in one step each agent uses a 'teleport' move to reach the location of a next task, which is then serviced in the second step. Clearly, this upper bound is an overestimation of the optimal value, since at each time-step new tasks appear at random locations and agents generally need more than one-step travel times to these tasks. However, as demonstrated in Table 4.3 the proposed approximations yield rewards relatively close to this upper bound. For instance, about 75% in the 6x6 world and about 70% in the "Office" world are achieved, indicating that our proposed approach demonstrates reasonable behavior even for these huge problems.

4.6 Summary

This chapter introduces SPATAPs, a general sub-class of MMDPs suitable for spatially distributed problems that a team of agents or robots needs to solve. Such tasks are characteristic of many realistic multi-robot systems, such as mobile sensor nets, distributed transportation and task assignment, multi-robot exploration, etc. To combat the complexity of general MMDP algorithms, we propose to use phase- and subjective approximations, and combine both to yield an efficient online planning method for SPATAPs. An extensive empirical evaluation shows that the proposed combination of approximation techniques yields near-optimal results for problem instantiations that we could solve optimally and further scales to much larger problems with thousands of states and joint actions.

In the following chapters, we will investigate how the proposed approach can be extended to larger and more realistic environments and eventually how it can be used to coordinate a team of real robots. This includes the application of approximate online MMDP planning methods such as Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006).

Chapter 5

Warehouse Commissioning as a Spatial Task Allocation Problem

In the previous chapters, we have developed methods for Multi-Robot System (MRS) to navigate in dynamic environments with uncertain localisation, and to perform actions without central coordination. In this chapter, we improve these techniques in order to be able to scale to real-world applications.

An important problem in *Industry 4.0* is multi-robot warehouse commissioning, where robots fetch and deliver items. A crucial aspect of warehouse commissioning is the coordination of workers over tasks. Currently, with human pickers, the task is tackled using pre-computed pick lists (Henn et al., 2012). These pick lists are generated centrally by the warehouse management system and then executed statically, which means that the pickers cannot be re-assigned when en-route.

In robotics, these kind of problems have also been phrased in the *Multi-Robot Task Allocation (MRTA)* (Gerkey and Mataric, 2003; Gerkey and Matarić, 2004) framework. In MRTA, the approaches assign a set of tasks to the robots, which then execute these from start to finish. A common solution to MRTA are auctioning based approaches, where each robot bids for the tasks according to their own evaluation. While well studied (Capitán et al., 2013; Dias et al., 2006; Gerkey and Mataric, 2002) and conceptually clear, these approaches have to two shortcomings: First, in a highly adaptive and changing environment, new tasks are likely to appear while the robots are on their way. The static assignments mean that the robots cannot respond in a timely manner. Second, these approaches typically require a centralised task allocation component, which may hinder scalability, flexibility and robustness.

In order to address these drawbacks, in the previous chapter, we presented a new formal framework of *Spatial Task Allocation Problems (SPATAPs)* that describes how a team of agents interacts with a dynamically changing set of tasks, where the tasks are spatially distributed over a set of locations in the environment. To recapitulate, SPATAPs form a sub-class of Multi-Agent Markov Decision Process (MMDP) (Boutilier, 1996) problems, which themselves are an extension of Markov Decision Processes

(MDPs) (Puterman, 1994) to multiple agents. These models have the advantage of providing principled solutions under uncertainty of action outcomes and provide clear definitions of optimality.

The SPATAP framework is fundamentally different from the MRTA framework. It enables planning on a much more fine-grained level: The robots re-plan their next action in every time-step, or as soon as the global state has changed. This, for example, might lead to a robot not taking the shortest path to an existing task, but a slightly longer path that brings it closer to locations where it is likely that important tasks appear in the near future. As such, the SPATAPs framework enables robots to anticipate task appearances in a way that is not possible with an MRTA approach.

As a SPATAP is an MMDP, it can be solved using MMDP solution methods. However, such methods are centralised. Moreover, these solutions do not scale well, due to the exponential explosion of state and action spaces when introducing more agents and tasks. An alternative approach is to tackle these problems using distributed online planning. That is, rather than centrally computing an optimal plan for all agents, each agent in parallel tries to compute a plan for itself. Such a decentralised approach has the benefits that it is robust against failure since there is no centralised planner, and flexible in that it allows to easily introduce more robots if required. In the previous chapter, we showed with an empirical study that such a decentralised approach for SPATAPs yields promising results in comparison to solving the MMDP and other baselines. Still, some limitations remain: the approach was demonstrated to scale to 60 locations, which —while still resulting in more than 6×10^{30} states (cf. Table 4.2b)— is far from the scalability desired in real-life warehouse commissioning tasks. Moreover, it relies on a number of rather ad-hoc approximations to keep the state space over which is actually planned small. Since Monte Carlo Tree Search (MCTS) (Kocsis and Szepesvári, 2006) methods allow for an effective treatment of problems with large numbers of states, this chapter investigates how such methods can be used to overcome the mentioned limitation.

In this chapter, we show how MCTS can be used in the SPATAP framework. We propose a number of computationally cheap rollout strategies that are specific for SPATAPs, and discuss how these relate to prior work (Sleight and Durfee, 2012; Capitán et al., 2013). We also investigate how certain domain-specific modifications of MCTS used in the Scotland Yard game (Nijssen and Winands, 2012) can be transferred to SPATAP.

Additionally, we extend the SPATAPs framework to facilitate more realistic modelling of commissioning warehouse tasks by including a drop-off point, i.e tasks have to delivered to a depot location. We further impose a maximum capacity constraint on the robots, such that they can only perform a limited amount of pick-ups before they have to return to the depot location to clear their load.

Extensive empirical evaluations show that our MCTS approach leads to significantly higher task performance, *especially for more complex problems*. Moreover, we also show that the approach scales better than the previous state of the art, demonstrating excellent performance on an 8-robot team servicing a warehouse comprised of over 200

locations.

The remainder of this chapter is structured as follows. Section 5.1 summarises the related work. Section 5.2 presents the problem description and the environment. We also recap the necessary theoretical background. In Section 5.3, we explain in detail how MCTS can be used in the SPATAP framework for warehouse commissioning. Section 5.4 presents the empirical results of the approaches. Section 5.5 concludes the chapter with a summary.

5.1 Related Work

The work of (Henn et al., 2012) relates to this research w.r.t. order picking and batching. Such approaches usually only generate static *pick lists*, which are then statically executed. They do not allow for online re-planning or changing the task allocations during execution. Furthermore, our approach can be run in a decentralised fashion.

Another well-known approach is the work done by KIVA (Wurman et al., 2008; Enright and Wurman, 2011). They designed a system in which multiple mobile driving robots bring complete shelves to the human pickers. On the shelves there are different items to be picked or stored upon. The approach has several shortcomings. The task allocation is done centrally and the mobile driving robots only execute the given orders. This means that there is no online optimisation possible after the orders have been assigned. Similarly, the robots do not decide on the same granularity as our approach; in the KIVA approach, the robots take the shortest path to the given target, while in our approach the robots make decisions on a much more fine-grain level like in which direction to drive.

Auction-based approaches can be found in (Amador et al., 2014; Choi et al., 2009). These do not address the sequential and changing nature of the tasks, since they assume tasks to be static. Also auctioning, generally relies heavily on communication, while our approach only needs the locations of the other agents.

Another related topic are general resource allocation problems (Wu and Durfee, 2010). The problem we address differs from that line of work in that we allow re-allocation at every time-step and consider spatially distributed tasks and travel times.

Some related work that is also using MCTS for solving task allocation problems is described in (Kartal et al., 2016). However, they deal with MRTA problems with time constraints, i.e. a static task set. A complete plan is computed beforehand in a centralised fashion and then executed, while with SPATAPs we are able to recompute the plan online at every change of the global state.

Generally, most competing approaches based on task allocation methods suffer from the problem that agents will not position themselves if no task is currently present, or rely on communication between the agents.

5.2 Warehouse Commissioning with multiple Robots

In warehouse commissioning, a team of agents needs to service a set of tasks distributed in the warehouse. These tasks are single item orders that have to be fetched from within the warehouse and brought back to the depot. Orders can be placed at any given time, so that new tasks appear over time.

A typical warehouse layout is a *rope ladder* (Seward, 2015), depicted in Figure 5.1. In this layout, the items are stored in shelves that are organised in aisles with some cross aisles. The shelves can have multiple compartments at different heights. This environment can be represented as a graph $\mathcal{G}$ with a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$ as overlaid in Figure 5.1. Each node represents a position in the warehouse where the agent can reach a number of storage compartments, on both sides of the aisle and at different heights. This limits the planning complexity without a loss in accuracy, since we assume that the agent can pick from different sides and heights equally well. One node is defined for the depot to which the orders have to be brought, marked in red in Figure 5.1. The agents have a limited inventory, i.e. they can pick and store a limited number of items, before they have to return to the depot to clear their load.

The agents can move along the edges in the graph. Each edge can have different costs, e.g. edges moving through free space could be quicker than moving along a shelf. There is also the possibility that movement is unsuccessful for the agent, due to wheel slip or other uncertainties.

Orders are mapped to the locations in the graph and appear as tasks at the corresponding nodes. We assume that the distribution of the orders is known, e.g. by analysing order histories. Therefore, we can model the probabilities of tasks appearing for each location. More specifically, each node has a given chance p that a task appears in the next time-step t, and p is known to the agents. The tasks can have different costs c.

5.2.1 A commissioning Spatial Task Allocation Problem

The commissioning problem can be modelled as a SPATAP, which is a special case of a *factored* MMDP as introduced in the previous chapter.

We expand the model with inventory states, leading to the following definition of a *commissioning SPATAP*.

Definition 5.1. A *commissioning SPATAP* is defined as a tuple $\langle \mathcal{D}, \mathcal{G}, \mathcal{I}, \mathcal{T}, \mathcal{S}, \mathcal{A}^M, \mathcal{A}^T, \mathcal{A}, P^M, P^T, R^M, R^T \rangle$, where
$\mathcal{D}$ is the set of n agents,
$\mathcal{G} = \langle \mathcal{N}, \mathcal{E} \rangle$ is a graph comprised of a set of nodes $\mathcal{N}$ and edges $\mathcal{E}$,
$\mathcal{I} = \{\mathfrak{I}_{empty}, \ldots, \mathfrak{I}_{full}\}$ describes the set of inventory states per agent,
$\mathcal{T} = \{\mathfrak{T}_0, \ldots, \mathfrak{T}_{|\mathcal{T}|}\}$ is the set of task states per location n_x,
$\mathcal{S}$ defines the set of states s, which can be factored as $s = \langle \lambda, \iota, \tau \rangle$, where λ and ι are the respective vectors of the locations and inventory states of all agents, and τ is the

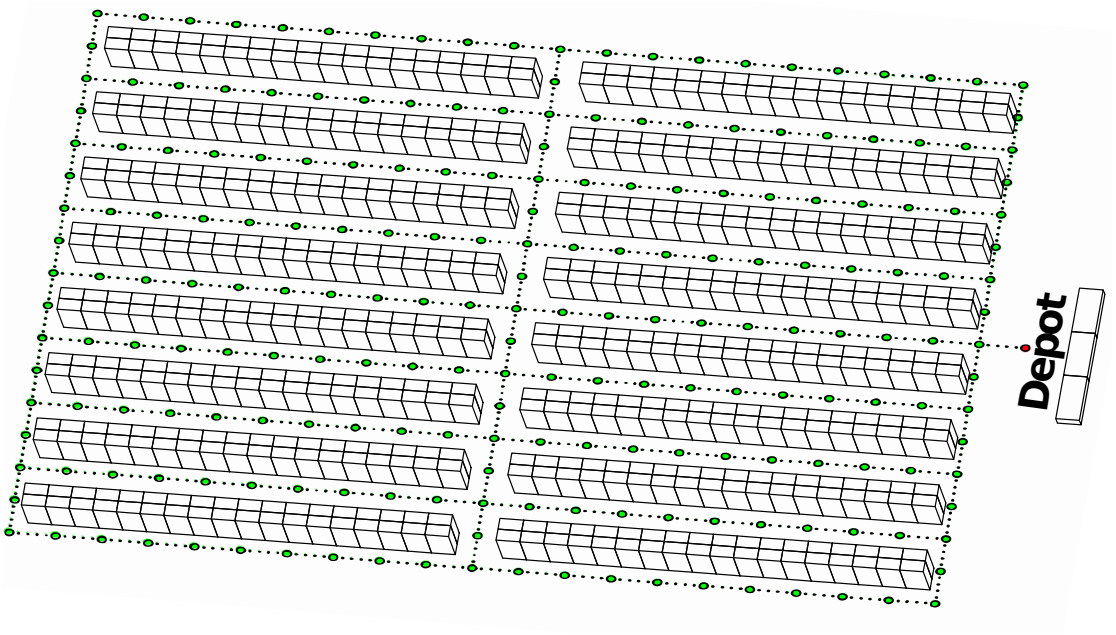


FIGURE 5.1: A rope ladder warehouse, with two blocks of shelves and a drop off depot. Overlaid is the resulting navigation graph from the warehouse model.

vector of the task status for all locations.
$\mathcal{A}^M$ and $\mathcal{A}^T$ are the sets of movement and task actions respectively, i.e. each agent i can choose an action $a_i \in \mathcal{A}^M \cup \mathcal{A}^T$ and $a = a_1 \dots a_n$ is the combined action for all agents.
$\mathcal{A}$ is the set of all possible joint actions.
$P^T = Pr(\tau', \iota' | \tau, \iota, \lambda, a)$ define the task and inventory transition probabilities, and
$P^M = Pr(\lambda' | \lambda, a)$ models the changes in agent locations.
$R^T(\tau, \lambda, \iota, a)$ and $R^M(\lambda, a)$ define the rewards for the tasks and the movements.

In the commissioning setting, the possible movement actions $\mathcal{A}^M$ are either staying or choosing any outgoing edge to go to the next node. The task related actions are defined as $\mathcal{A}^T = \{PERFORM\_TASK,\ CLEAR\_LOAD\}$. The task transition function models the execution of a task, e.g., when an agent performs a task at a node and the inventory of the agent is not yet exceeded, then the task state of the node and the inventory of the agent are updated, and it also models the exogenous events of new tasks appearing, i.e. by orders being placed.

While using the factored components enables us to relatively easily simulate the environment, already enumerating all possible states may prove difficult, since the resulting state space is equal to

$$|\mathcal{S}| = |\mathcal{N}|^{|\mathcal{D}|} \cdot |\mathcal{I}|^{|\mathcal{D}|} \cdot |\mathcal{T}|^{|\mathcal{N}|}.$$

Thus, the size depends exponentially on the number of agents and the number of locations.

Algorithm 7: Updated SimulateQ function for Sparse-UCT

```
function SimulateQ(s, q_node, a)
    if EnoughSamples(q_node, a) then
        s', r ← RandomChild(q_node)
    else
        s', r ← SimulationStep(s, a)
    d ← d + 1
    if Visited(s') then
        r' ← SimulateV(s')
    else
        r' ← Rollout(s')
    d ← d − 1
    r ← r + γ * r'
    return r
```

5.3 Monte Carlo Tree Search for SPATAPs

A shortcoming of the EFWD approach as introduced in the previous chapter, is that limiting the state space can lead to unwanted outcomes. For instance, using a phase-myopic approach that only considers the nodes with currently active tasks might lead to the agents not moving at all if there is currently no task present. This could lead to large losses in performance compared to already moving towards the most likely positions where the next tasks may appear.

In this chapter, we adopt the basic principle of distributed planning, including the need to (approximately) model other agents, but we seek a more principled way of dealing with the complexity of the individual planning tasks. In particular, rather than limiting the size of the state space of these problems with ad-hoc approximations, we build on principled methods to deal with large state spaces. We build upon sample-based planning methods whose performance is *independent* of the size of the state space (Kearns et al., 2002). We base our work on MCTS as introduced in Section 2.2.4 with UCT (Kocsis and Szepesvári, 2006), which uses UCB1 (Auer et al., 2002) to select actions inside the tree. UCT, however, is not directly amenable to distributed planning for SPATAPs. There are a number of challenges that need to be overcome: First, there is an issue with the extremely large number of states we want to consider. Second, it turns out that it can be important to incentivize agents to do tasks themselves, rather than relying on other agents to do them. Finally, a major difficulty is how the planning agent can predict the behaviour of other agents. Previous work based such predictions on full solutions of the 'self-absorbed' MDP, but this is not feasible for the problem sizes we want to consider, and is too slow for application in a MCTS rollout policy. In the remainder of this section, we propose a number of techniques addressing these issues.

5.3.1 Dealing with huge State Spaces

While MCTS methods can deal with fairly large state spaces, huge state spaces are problematic. If the number of states reachable from a given state by a given action is very large for example, it can lead to the algorithm always being 'outside the tree' and never building up meaningful statistics. This is a problem in SPATAPs: due to tasks appearing at random nodes, it is very unlikely in our problem setting that choosing the same action from the same state would ever lead to the same successor state twice.

To deal with this problem, we propose to use Sparse UCT (Bjarnason et al., 2009) for SPATAPs, which builds upon 'Sparse Sampling', an online planning method whose performance is *independent* of the size of the state space (Kearns et al., 2002). The main idea behind Sparse Sampling is that it is sufficient to sample, for each visited node and action, only some constant number w of successor nodes. We therefore keep track of how many successor states have already been created for the same state and action, and if the sample width w is exceeded, we return a random existing successor state instead of sampling a new one. Algorithm 7 shows the adaption of the `SimulateQ` function to incorporate the changes of Sparse UCT.

5.3.2 Incentivizing Agents

Accurately modelling the behaviour of the other agents is a fundamentally difficult problem (which we treat in the next sub-section), and it is unlikely that an agent will be able to make perfect predictions about what tasks will be addressed by the other agents. However, incorrectly assuming that an important task will be addressed by a team mate can lead to high costs. Therefore, in the face of such uncertainties, it might be good if agents have a slight preference to do tasks themselves.

The inspiration from this comes from the cooperative game of Scotland Yard, where it has been shown that the planning agent sometimes relies too much on other agents fulfilling their task of catching the 'hider' (Nijssen and Winands, 2012). The authors propose to discount the value of a successful rollout by $r \in [0, 1]$ if a different agent than the planning agent caught the hider. This idea cannot be directly translated to our approach, since we have no definition of 'successful rollout'. However, since we also observed the behaviour that the planning agents were relying too much on the other agents to perform a task, we propose a *do-it-yourself (DIY)* bonus if a task is performed by the planning agent. The action *PERFORM_TASK* is awarded an additional bonus $DIY_r \in [0, 1]$ if it is performed by the planning agent. If the bonus is too small, especially in symmetric configurations, there can be a tendency for no agent to do the task; if it is too high, both agents tend to try to perform the task.

5.3.3 Modelling Other Agents: Greedy Rollout Policies

Good predictions of team mates are critical for the success of any distributed planning approach. For an approach based on MCTS this is even more difficult, since action

predictions for other agents are needed in every step of every simulation. This means that the predictions should not only be sufficiently accurate, but also computationally cheap.

For this purpose we introduce three greedy heuristics for SPATAPs. We do not only use them to predict the actions of other agents (both 'inside' and 'outside' of the tree), we also use them as the rollout policy for the planning agent (only 'outside' the tree), which performs much better than a random rollout policy as we show empirically.

All three approaches are based on heuristic valuations of how much value each agent i can generate for each task location n_x. The evaluation function we use is defined as follows:

$$\texttt{NV}(n_x, i) = \begin{cases} -\infty & \text{if } \tau_{n_x} = \mathfrak{T}_{empty} \\ \frac{\texttt{TV}(\tau_{n_x}, \iota_i)}{\texttt{dist}(\lambda_i, n_x)} & \text{otherwise} \end{cases} \tag{5.1}$$

where $\mathfrak{T}_{empty}$ means that there is no task present, $\texttt{dist}(\lambda_i, n_x)$ is the length of the shortest path from the agent i to the node n_x and $\texttt{TV}(\tau_{n_x}, \iota_i)$ denotes an evaluation of the task status at node τ_{n_x}, given the current inventory status of the agent ι_i. In our case this is the largest sum of the cost values of the tasks at the node that can still fit in the inventory of the agent, i.e. we keep adding the task with the highest available cost until either the inventory is full, or there is no task left. This function TV can be used to influence the agents' behaviours. It could for instance also include other factors such as the time the task is already active, if older tasks should be valued higher. As NV is a reward function, we aim to maximise its value.

For all three heuristics, we assume that if an agent is at its capacity limit, it will choose the shortest path to the depot and unload. These agents are also not further considered in the planning process, until their inventory is empty again. In the following the three heuristics will be explained in more detail.

1. Greedy with Social Law

   A very simple heuristic is for all agents i to always move towards the node that has the highest evaluation according to our NV function, i.e. $a_i = \texttt{GoTo}(\max_{n_x \in \mathcal{N}} \texttt{NV}(n_x, i), i)$, where the function $\texttt{GoTo}(n_x, i)$ returns the next action on the shortest path of agent i to node n_x. If the agent is already at that node, it returns the action *PERFORM_TASK*.

   In order to overcome symmetries, we can apply a social law similarly to the previous chapter: If two agents have the same node with the highest reward, the one with the higher id will go to the best evaluated node, while the next ranked agents will go to the next ranked node. This heuristic will be used as baseline for our evaluation.

Algorithm 8: Reverse Greedy

Input : s: state $< \lambda, \iota, \tau >$
Output: a: actions for all agent

Let V be a vector of size $|\mathcal{D}|$
Assign $V[i] \leftarrow \infty \; \forall i \in \mathcal{D}$
foreach $n_x \in \mathcal{N}$ **do**
  $v_{best} \leftarrow \max_{i \in \mathcal{D}} \texttt{NV}(n_x, i)$
  $a_{best} \leftarrow \text{argmax}_{i \in \mathcal{D}} \texttt{NV}(n_x, i)$
  if $v_{best} > V[a_{best}]$ **then**
    $a[a_{best}] \leftarrow \texttt{GoTo}(n_x, a_{best})$
    $V[a_{best}] \leftarrow v_{best}$
return a

2. Reverse Greedy Allocation

   The general idea behind the Reverse Greedy heuristic is to look at the problem from the perspective of the nodes. The idea is that each location in the graph is assigned to the agent that has the best evaluation for this node. This is also comparable to a partition organisation as in (Sleight and Durfee, 2012), since all the tasks are distributed between the robots according to their evaluations. However, this approach adaptively changes the partition according to the changing locations of the agents and tasks.

   Algorithm 8 shows pseudocode for this policy. The `argmax` function returns the agent that corresponds to the maximum node value v_{best}. This approach intrinsically takes care of a social law if the agents are always iterated in the same order by the `Index` function, e.g. decreasing by id.

3. Iterative Greedy Allocation

   For the final rollout policy that we propose, we can take inspiration from auctioning approaches (as commonly used in MRTA, i.e. in (Capitán et al., 2013)). We evaluate all locations for all agents and iteratively assign the currently best evaluated location to the highest ranked agent. This is very similar to Sequential Single Item (SSI) (Algorithm 4) as presented in Chapter 2. More specifically, we compute `NV` for all agents and all locations. The agent that has the node with the best evaluation over all agents and locations, gets the location assigned and both, the agent and location are removed for the future allocations. In the next iteration, the agent which has the best evaluation for the remaining node is selected and so on. Algorithm 9 shows pseudocode for this approach. The function $\texttt{GetBest}(V, i)$ returns the currently best evaluated location for agent i and its value.

Note that both Reverse and Iterative Greedy are centralised algorithms that suggest actions for all agents for a given state. However, these algorithms can be run decentralised on the robots, which is possible, since the global state is known. During the planning

Algorithm 9: Iterative Greedy

Input : s: state $< \lambda, \iota, \tau >$
Output: a: actions for all agent

Let V be a matrix of size $|\mathcal{D}||\mathcal{N}|$
$V[i][n_x] \leftarrow$ `NV(`n_x, i`)` $\forall n_x \in \mathcal{N} \; \forall \; i \in \mathcal{D}$
$A_{ass} \leftarrow \emptyset$; $N_{ass} \leftarrow \emptyset$
for *iteration* $x < |\mathcal{D}|$ **do**
  $n_{best} \leftarrow$ None; $a_{best} \leftarrow -1$; $v_{best} \leftarrow -\infty$
  foreach *Agent* $i \in \mathcal{D}$ *and* $i \notin A_{ass}$ **do**
    $n_{sel}, v_{sel} \leftarrow$ `GetBest(`V, i`)`
    while $n_{sel} \in N_{ass}$ **do**
      $V[i][n_{sel}] = -\infty$
      $n_{sel}, v_{sel} \leftarrow$ `GetBest(`V, i`)`
    if $v_{sel} > v_{best}$ **then**
      $v_{best} = v_{sel}$;$a_{best} = i$; $n_{best} = n_x$
  $a[a_{best}] \leftarrow$ `GoTo(`n_x, a_{best}`)`
  $A_{ass} \cup a_{best}$; $N_{ass} \cup n_{best}$
return a

process, each agent needs to predict actions for all other agents, thus these heuristics are an efficient way to calculate these estimated actions all at once.

More specifically, during the UCT simulations: when 'inside' the tree, the planning agent overrides the action prescribed by these algorithms with the action prescribed by UCB1. In the rollout phase (i.e., outside the tree) all agents follow the prescribed actions. Additionally, these heuristics serve as baselines. In this case, they are used without any MCTS search. Each agent computes the heuristic until his action is assigned and acts accordingly.

5.4 Experiments and Results

To empirically evaluate our proposed MCTS approach, we have implemented a warehouse simulation and the proposed rollout policies in ROS (Quigley et al., 2009). Each agent is simulated independently. This is realised by running each agent's planning approach in a different process. Only the global state (λ, ι, τ) is communicated from the simulator to the agents, and there is no direct communication between agents. The depot location is marked in red and the normal task nodes in green. The agents always start at the depot. Movement actions have a 90% chance of succeeding, which simulates the uncertainty in the real world, e.g. due to wheel slip or other sensor noise.

The probability of tasks appearing at a node is drawn randomly from a set of three probabilities ($p_{low} = \frac{0.2}{|\mathcal{N}|}, p_{mid} = \frac{0.4}{|\mathcal{N}|}, p_{high} = \frac{1}{|\mathcal{N}|}$). This simulates that certain areas in the warehouse may store items that are ordered at a higher frequency and vice-versa.

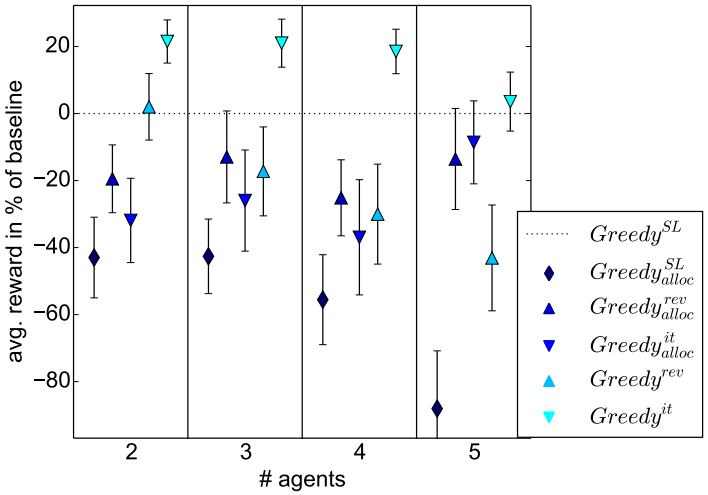


FIGURE 5.2: Fixed vs online (re)-allocation for different policies using the world *warehouse-small* with 30 nodes and different number of agents. The whiskers show the 95% confidence intervals.

Each task that appears can have a different priority, i.e. higher costs of not servicing this task. These are sampled according from the set of $(c_1 = 1, c_2 = 2, c_3 = 5)$ with probabilities $(p_{c_1} = 0.8, p_{c_2} = 0.1, p_{c_3} = 0.1)$.

The seeds for creating the probability distribution and for tasks appearing are synchronised between the different runs, such that for all policies, the same number of tasks appear at the same locations during the run. We run 30 simulations of 100 time-steps each, unless reported differently. As a baseline we use the proposed rollout policy *greedy with social law* ($Greedy^{SL}$) without any MCTS search. We compare our approach against the *EFWD* approach as introduced in Chapter 4 with the k-SP-MDP approximation and $k = 5$ nearest tasks, and additionally, we compare against using the two heuristics, Reverse Greedy ($Greedy^{rev}$) and Iterative Greedy ($Greedy^{it}$), without any MCTS search. For this evaluation, each planning agents computes the centralised policy until its own action is assigned.

For our approach, the MCTS search runs 20,000 simulations up to a depth of 60 steps, while choosing the actions during the rollouts by one of the rollout policies introduced in the previous section, i.e. Greedy with Social Law ($MCTS^{SL}_{Greedy}$), Reverse Greedy ($MCTS^{rev}_{Greedy}$) and Iterative Greedy ($MCTS^{it}_{Greedy}$) or purely random selection ($MCTS_{random}$). We implemented all rollouts in an ε-greedy manner, i.e. uniformly random actions are performed with a probability of $\varepsilon = 0.05$. These settings were determined empirically.

5.4.1 Fixed allocation vs online re-allocation

We compared the heuristics using fixed allocation, i.e. the robots choose their tasks and do not change it until it is performed, against online re-allocation, meaning that the robots re-plan at every time-step. Thus, the fixed allocation methods (defined as

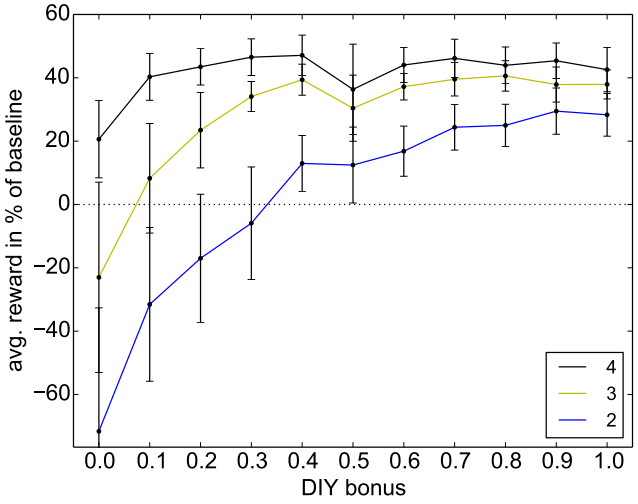


FIGURE 5.3: DIY Bonus for different policies using the world *warehouse-small* with 30 nodes and different number of agents. The whiskers show the 95% confidence intervals.

$Greedy^{SL}_{alloc}$, $Greedy^{rev}_{alloc}$ and $Greedy^{it}_{alloc}$) simulate behaviour as following the MRTA framework, while online re-allocation is according to the SPATAPs definition. The results are presented in Figure 5.2, where the whiskers show 95% confidence intervals. $Greedy^{it}$ outperforms the baseline $Greedy^{SL}$ and all other approaches with up to four significantly, while with five agents the the fixed allocation methods, $Greedy^{rev}_{alloc}$ and $Greedy^{it}_{alloc}$ are catching up.

When using fixed allocations, the two proposed heuristics, Reverse and Iterative Greedy, are almost on par. However, using $Greedy^{rev}$ with online planning actually decreases its performance, especially with more agents. The decreasing performance can be explained by the nature of the simulation. As the agents all start at the depot, all potential tasks are allocated only to the highest ranked agent. Thus the agents will not spread out. When the allocation is fixed, the agents keep moving out as soon as a task is assigned, thus they spread more making the partitioning of the $Greedy^{rev}$ more effective. Additionally, that all methods (with the exception of $Greedy^{rev}$) move closer together with more agents, is the result of having relatively less tasks per agent. Thus the effect of online planning is smaller. With larger worlds, thus relatively more tasks, this effect is less.

To summarise, the online re-allocation yields a big advantage especially for $Greedy^{it}$ and the $Greedy^{SL}$ baseline, while for $Greedy^{rev}$ is actually a disadvantage.

DIY-Bonus

To evaluate which value for DIY_r yields the best performance, we run the simulation for different number of agents and increasing values of DIY_r. Figure 5.3 shows the resulting rewards with their 95% confidence intervals. As can be seen, the value of DIY_r has a high influence on the reward, especially for 2 agents. With only 2 agents, there

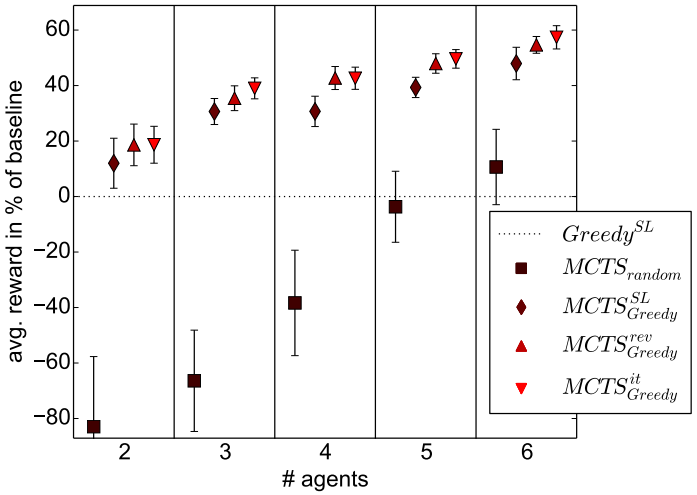


FIGURE 5.4: Different rollout heuristics for different policies using the world *warehouse-small* with 30 nodes and different number of agents. The whiskers show the 95% confidence intervals.

are more situations in which there is a tie for a certain location, so it is better when both head towards it, instead of no one. With more agents these situations occur less frequently. For the remaining experiments we set the DIY bonus to 0.7 as a trade-off, since especially in smaller worlds we experienced sub-optimal behaviour due to multiple robots moving towards the same task location.

5.4.2 Different rollout policies

When comparing the different rollout policies, the results in Figure 5.4 show that the $MCTS^{SL}_{Greedy}$ outperforms the $Greedy^{SL}$ baseline significantly. Thus adding MCTS search to the heuristics significantly improves the performance. However, MCTS in itself with using the random rollout policy ($MCTS_{random}$), performs less well than the baseline, showing that the rollout policies have a large influence on the performance.

Using Iterative and Reverse Greedy for the rollouts yields a significantly higher performance than the other policies, especially for more agents. Both of these strategies already take care of some task allocation, which helps to improve the action selection. Therefore, we focus on these two approaches for further evaluation.

5.4.3 Planning times

To investigate the planning times, we randomly sampled 200 different states and averaged the time it takes to plan in these states. We also included the previous *EFWD* approach. The results are summarised in Figure 5.5. We can see that the search times of all approaches increase roughly linearly with the number of agents. However, while the MCTS-based approaches have a very consistent planning time, the *EFWD* approach varies greatly. This is due to the k-nearest task approximation. When there is no task

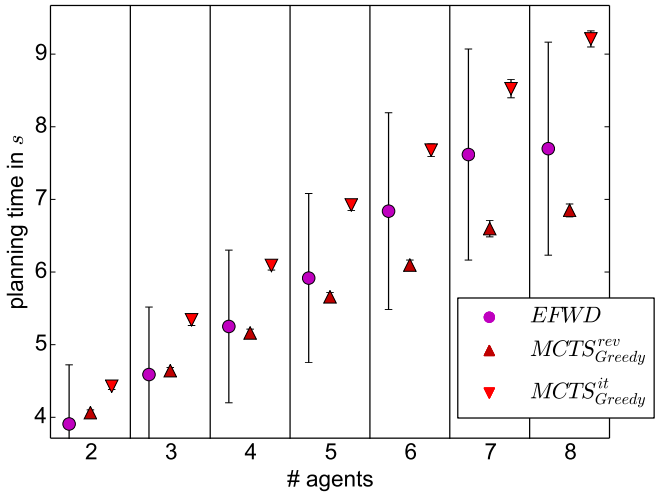
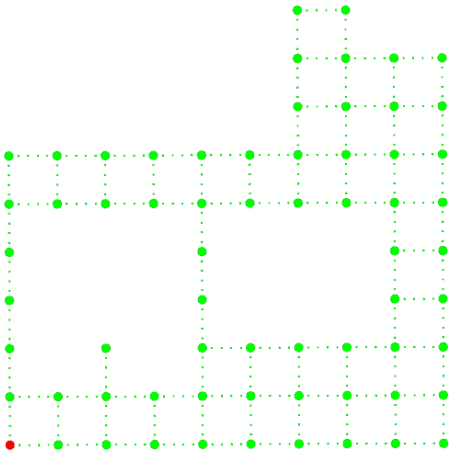


FIGURE 5.5: Planning times for different policies using the world *warehouse-medium* with 66 nodes and different number of agents. The whiskers show the 95% confidence intervals.

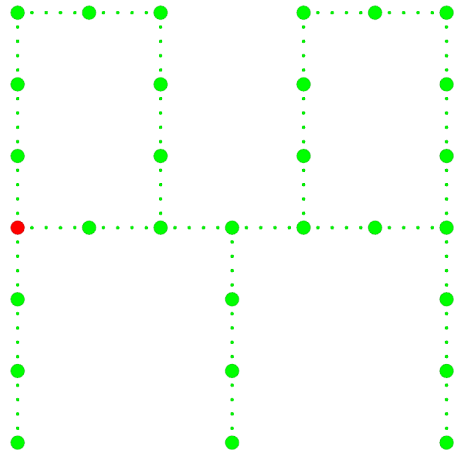
active, the planning time is nearly 0. However, as soon as there are one or more tasks active, the planning time increases greatly. We can also see that $MCTS^{rev}_{Greedy}$ is significantly faster than $MCTS^{it}_{Greedy}$, since it does not need to iteratively assign the tasks after the evaluation.

5.4.4 Limited vs unlimited capacities

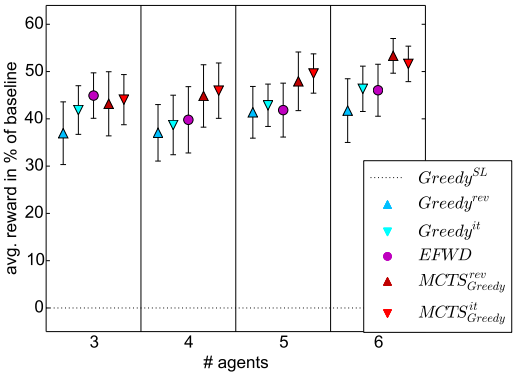
We have evaluated the approaches in settings where the agents have unlimited capacity and the tasks appear uniformly distributed over the warehouse. Essentially, when the agents have unlimited capacities, the depot node is obsolete, since the agents will never return to unload. Figure 5.6 shows the result for the *warehouse-small* environment and the *office* environment (the *office* is the same as in the previous chapter). In comparison, with unlimited capacities (cf. Figure 5.6d and 5.6c), the resulting performances are closer to one another than with limited capacities (cf. Figure 5.6f and 5.6e). The MCTS based approaches still outperform all other approaches. The proposed heuristics without MCTS ($Greedy^{rev}$ and $Greedy^{it}$) perform as good as the *EFWD* approach. Most significantly, the $Greedy^{rev}$ policy performs almost as good as the $Greedy^{it}$ policy, which is in great contrast to the limited setting, where it yields even less reward than the $Greedy^{SL}$ baseline. This can be explained by the structure of the unlimited setting. In principle, the adaptive partitioning of the $Greedy^{rev}$ should spread the agents out, since tasks are appearing and the robots are moving out. After a while, all agents are spread out and assigning tasks based on their locations works well. However in the limited settings, the agents always need to return back to the depot, which mitigates the effect of the partitioning, since the agents get clustered at the depot. This problem is overcome when adding the MCTS search on top of the rollouts. The search helps all agents to start spreading out, even in the limited setting.



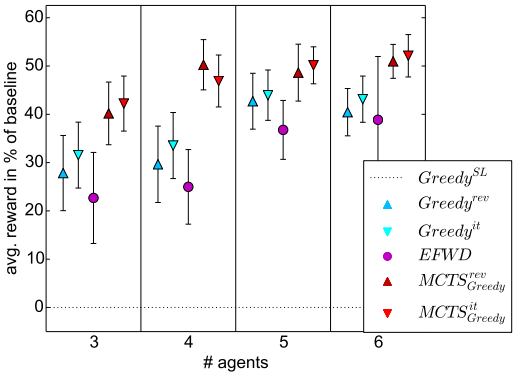
(a) *office-world*, #nodes=66



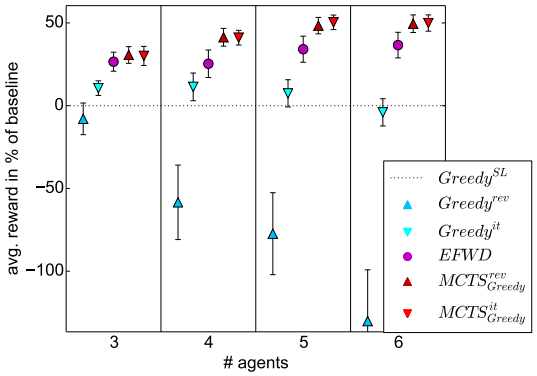
(b) *warehouse-small*, #nodes=30



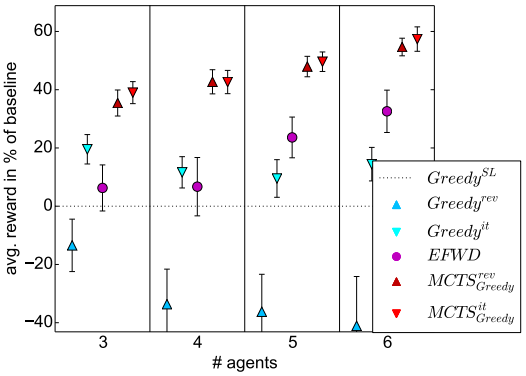
(c) *office-world*, unlimited capacity



(d) *warehouse-small*, unlimited capacity



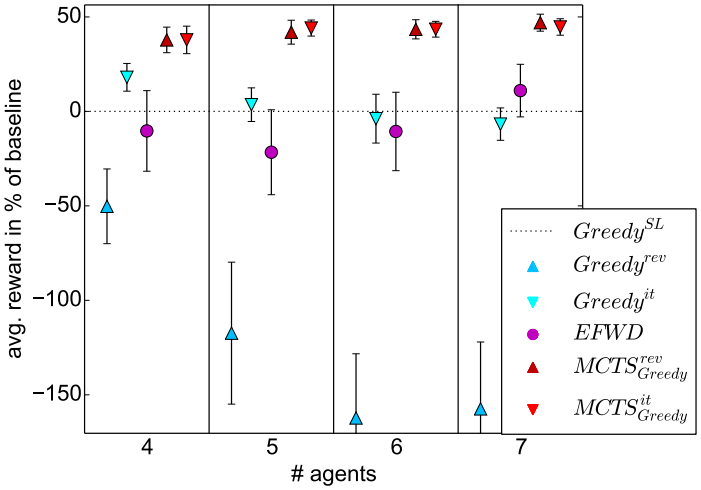
(e) *office-world*, limited capacity



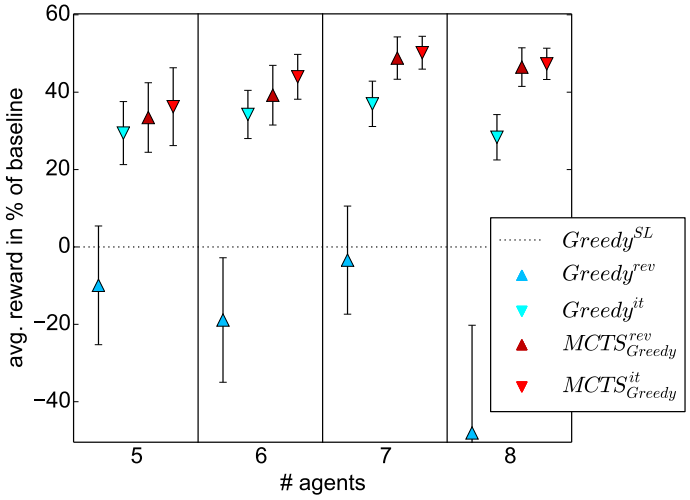
(f) *warehouse-small*, limited capacity

FIGURE 5.6: Comparing the different approaches in the worlds *office* (a,c,e) and *warehouse-small* (b,d,f) with infinite capacities and uniformly distributed task appearances and with limited capacities and distributed task appearances. The whiskers show the 95% confidence intervals.

EFWD performs much better in the *office-world* in comparison to the *warehouse-small*. This is most likely due to the structure of the worlds. The *office-world* is much more inter-connected, with almost no dead ends. Since *EFWD* does no positioning when there are no tasks present, it helps that the average shortest path length between nodes is a lot shorter in *office-world*.



(a) *warehouse-medium*, #nodes=66



(b) *warehouse-large*, #nodes=214

FIGURE 5.7: Comparing the different approaches in larger warehouse sizes and with different numbers of agents. The whiskers show the 95% confidence intervals.

To conclude, while the presented heuristics work really well in simple environments already by themselves, adding MCTS search still improves their performance. *EFWD* yields good performance in highly connected worlds.

5.4.5 Larger warehouses

Additionally, we compared the different approaches in two larger sized warehouse models, *warehouse-medium* with $n = 66$ (cf. Figure 5.8) and *warehouse-large* with $n = 214$ (cf. Figure 5.1). For the large warehouse, we increased the number of simulated steps to 250 and the number of repetitions was decreased to 15.

EFWD was not able to complete any run in *warehouse-large* due excessive planning times, i.e. more than 1000 seconds for one step. The results are shown in Figure 5.7.

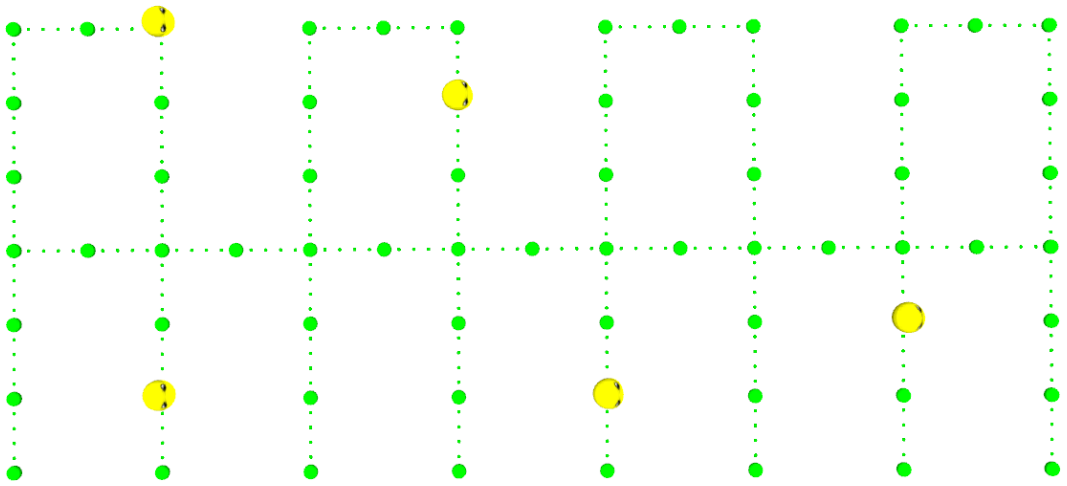


FIGURE 5.8: Positioning of $MCTS^{rev}_{Greedy}$ in the world *warehouse-medium* after 50 steps. The appearance of tasks is disabled, but agents' world model still assumes that new tasks appear. All agents started in the top left corner. Other approaches like *EFWD* and the heuristics without search would not have moved at all.

EFWD shows a generally increasing performance for more agents, but the relative rewards against the baseline are varying. Only for larger numbers of agents it can outperform the baseline and standalone rollout strategies. We can see that the two MCTS approaches perform nearly identically for *warehouse-small* and *warehouse-medium*. The good performance of $Greedy^{it}$ in *warehouse-large* is due to the high chance that there are many tasks active in a larger world. Therefore, iteratively assigning the best tasks yields a very good result. As soon as the number of agents increases however, MCTS search improves the result again, since there are relatively fewer tasks to distribute, and positioning becomes more important again.

5.4.6 Positioning

To show the effect of positioning, we let $MCTS^{rev}_{Greedy}$ run for 50 steps, while we disabled the appearance of new tasks assumed by the agents' world model. Figure 5.8 shows that the robots are nicely spread out. This is in stark contrast when using the heuristics without any MCTS search and also the *EFWD* approach. These remain in the same position if no tasks is present.

5.5 Summary

In this chapter, we look at the problem of multi-robot warehouse commissioning. The problem is modelled as commissioning SPATAP, which includes the capacity constraints of robots and a drop-off point. Previous solution methods do not provide the necessary scalability for large problem sizes. As a solution, we introduced a MCTS approach for these kind of problems.

Three different greedy policies are introduced, which can be used to predict the other agents' behaviours and also for the planning agent during the roll-outs. By combining

sparse UCT with these greedy heuristics, we are able to plan online in large problem sizes with up to 200 locations and a 8-robot team.

Our empirical evaluation shows that we can greatly improve the current state-of-the-art, while also being able to solve much larger problems.

In the next chapter, we will investigate on deploying the approach on real robots. We are setting up a multi-robot approach that is inspired by the RoboCup@Work (Kraetzschmar et al., 2014) competition. Multiple robots have to perform pick tasks in the environment and bring them to a common depot node.

Chapter 6

Real World Multi-Robot Warehouse Commissioning

In this chapter, we combine the previous work introduced in Chapter 3 with the work in Chapter 4 and Chapter 5 to achieve a real world decentralised Multi-Robot System (MRS). In the previous chapters, it was assumed that the robots are able to share edges and nodes. While using the work as presented in Chapter 3 enables the robots to navigate freely in the environment, there is a physical restriction the robots cannot be at the same location at the same time. This has to be taken into account while planning. Additionally, we need to deal with the asynchronous aspects of the robots' movements. More specifically, picking up an object takes considerably more time than moving to another location.

We implement small scale warehouse environment in which a MRS is employed. Up to three robots have to fulfil tasks in the environment, where a task is bringing items to a common depot node. These tasks appear online over time with different priorities. Thus, the coordination between the robots is an important factor to ensure that all tasks are fulfilled in a timely manner.

The setting is inspired by the RoboCup@Work (Kraetzschmar et al., 2014) and the RoCKIn@Work (S. Schneider et al., 2015) competitions. In these competitions, a single robot has to fetch and deliver items in a small scale industrial setting. The focus is on the capabilities of the individual robot like object detection, object manipulation, and navigation.

We present the necessary preliminary work to implement such a real world warehouse commissioning system. A reliable object recognition method is introduced and a model for the inverse kinematics of the youBot arm is presented.

We are using up to three youBots concurrently, which all are running the same algorithm onboard. The only communication between the robots is for the collision avoidance as presented in Chapter 3 and there is a central warehouse management system, which notifies the youBots when the tasks are updated, i.e. a new order has come in or another robot has picked an object.

We compare two approaches, namely using the iterative Greedy algorithm and combining it with Monte Carlo Tree Search (MCTS) as explained in the previous chapter. Our results show that the system is able to perform this complex task reliably.

The remainder of this chapter is structured as follows. Section 6.1 introduces the robot competitions in more detail and summarises the related work. Section 6.2 details the problem description and the approach. Section 6.4 presents the empirical results of the approaches. Section 6.5 concludes the chapter with a summary.

6.1 Robot Competitions and Related Work

There exist a few competitions that aim to stimulate research in the field of MRS and Multi-Agent Systems (MAS). The most well known competition is RoboCup, which originally held mainly different robot soccer competitions with the aim to defeat the human world champion by 2050 (Kitano et al., 1995).

In recent years, multiple other competitions have been added, whereas for this research the most interesting is the RoboCup@Work (Kraetzschmar et al., 2014) competition. Another similar competition to RoboCup@Work, is the European funded RoCKIn competition which was held in 2014 and 2015 and also featured an @Work competition (S. Schneider et al., 2015). In this competition the focus was to design a benchmark testbed for the Factory of the Future (FoF).

In the @Work competition, the goal is for a single robot to fulfil pick and place orders in a small-scale industrial warehouse-like environment. The focus is on the individual capabilities of the robot like object detection, object manipulation, and navigation.

The competition is currently run with a single robot only, while there are plans to have multiple teams competing at the same time. There is no standard platform required for this competition, but most teams use a youBot with some modifications, for instance the smARTLab@Work team competed with the youBot presented in Appendix A.1.

6.1.1 Related Work

There is some related work showing MRS in the real world. A notable work is (E. Schneider et al., 2016), in which a total of 192 real world runs were performed with three Turtlebots together with 960 simulation runs. In the work, various auctioning approaches are evaluated on different dimensions according to the Multi-Robot Task Allocation (MRTA) taxonomy. However, the act of performing a tasks is simulated by the arrival of the robot. Likewise, the auctioning is performed by a central auctioneer, while in our approach, we show a completely decentralised system, which actually picks and returns the items.

In (Niemueller et al., 2014), the Carologistics team competing in the Logistics League sponsored by Festo (Niemueller et al., 2016) describes their decisive factors for winning the 2014 competition. They show that they are using an elaborate central planning

component, which decides the tasks for all the robots. The centralised planning is different from our approach, which employs decentralised planning on robot level.

In another work, in (Alonso-Mora et al., 2015a), they show how two youBots collaborate together to transport an object and navigate within a work space with obstacles. They are relying on an external positioning system and the focus of the task is to transport the object, while keeping the formation. In our approach, we use Convex Outline Collision Avoidance under Localization Uncertainty (COCALU) as presented in Chapter 3, which does not need any external localisation. Also, our tasks are dynamically changing in the environment, while in the presented research there is only the single transportation task to fulfil.

The work in (Khandelwal and Stone, 2017) shows a human guidance system, in which a centralised controller makes use of various robots to provide navigation guidance to humans. A MCTS and a heuristic approach is used to solve an Markov Decision Process (MDP) formulation of the problem. While the general approach using MCTS, heuristics and an MDP formulation are closely related to our approach, the centralised component is in stark contrast to our decentralised approach.

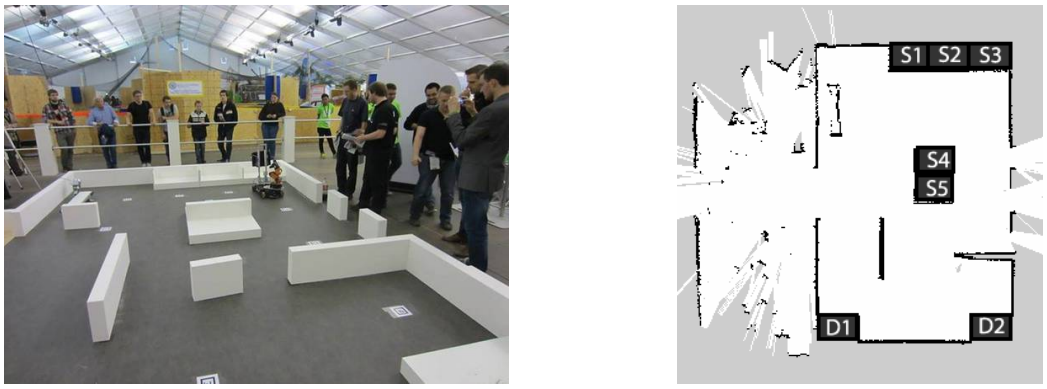
In (Amato et al., 2015), so-called *macro-actions* with Dec-POMDPs are introduced. The macro-actions define high-level actions, i.e. navigating to a different waypoint or grasping an object. In (Amato et al., 2016), a multi-robot implementation of this *macro actions Dec-POMDP* is shown for a bartender situated. Up to two *waiter robots* have to check three different rooms to see if the customers want to order drinks. If drinks are ordered, the waiters have to fetch a drink from the bartender robot and deliver it to the customers. As mentioned before, the Dec-POMDP framework assumes a partly observable state and another advantage of this approach is that it does not necessarily need communication. In this implementation, the robots plan a policy offline and during run-time only execute this policy based on their observations and previous actions. In contrast, our implementation is able to re-plan online as soon as a new global state is observed.

To summarise, while there are several competitions aiming to foster research in the direction MRS, there is limited published research in this field which can be directly compared with the proposed system.

6.2 Problem Description and Environment

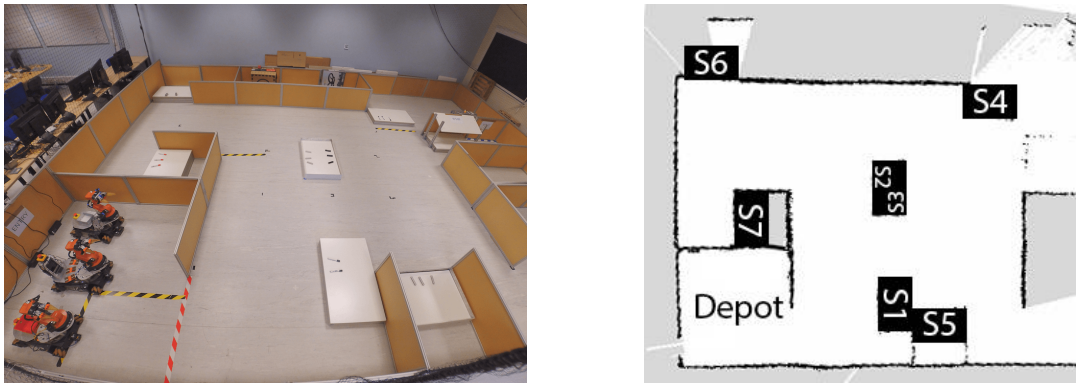
We assume a warehouse commissioning problem as introduced in Chapter 5. In more detail, there is a team of robots that has the task to fetch objects from within the warehouse on-demand. Thus, the orders can be placed at any point in time, according to a model of the previous order frequencies, which is known to the robots.

The objects are distributed on various platforms, also called *workstations*, and the robots have a common map of the environment including the locations of these workstations. However, the robots do not know the exact location of the objects. Hence, it



(a) (b)

FIGURE 6.1: (a) RoboCup@Work arena from the competition in 2014 with extra static obstacles. (b) Map of the arena. Annotated with service areas. D1 and D2 are the depot areas and S1-S5 are the possible pickup locations.



(a) (b)

FIGURE 6.2: (a) The test environment in our lab with the three youBots located in the depot area. (b) Map of the arena. The depot area and the possible pickup locations S1-S7 are annotated.

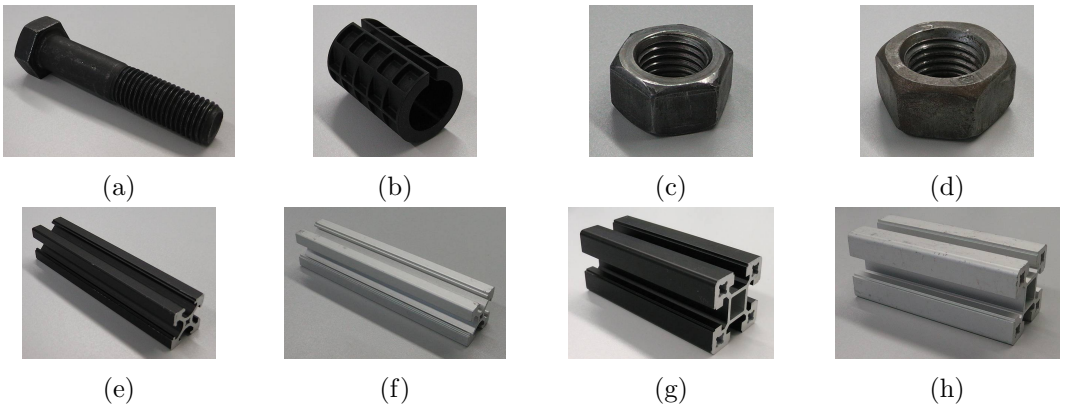
is necessary to search the platform for the correct objects. The robots are allowed to move freely within the environment.

The solution should be decentralised in order to be robust against single point of failures and the planning has to be done online in order to cope with additional tasks that appear.

In this section we will present the environment and assumptions that are used for our approach. The problem we are trying to solve is described and the necessary requirements for the solution are sketched.

The environment is inspired by the RoboCup@Work arena. In the arena are several so-called *service areas*, i.e. manipulation platforms, on which the objects can be grasped and placed. Figure 6.1a shows a picture of the 2014 RoboCup@Work arena and Figure 6.1b shows the corresponding annotated map that is used for navigation.

We changed the environment slightly to fit in our lab. There is a single depot area located in the lower left (cf. Figure 6.2), and there are seven pickup *service areas* (S1-S7)



(a) (b) (c) (d)

(e) (f) (g) (h)

FIGURE 6.3: Original RoboCup@Work Manipulation objects. (a) M20 Bolt (M20_100), (b) R20, (c) M20 Nut, (d) M30 Nut, (e) F20 Black (F20_20_B), (f) F20 Grey (F20_20_G), S40 Black (S40_40_B), S40 Grey (S40_40_G).

(a) (b) (c) (d) (e)

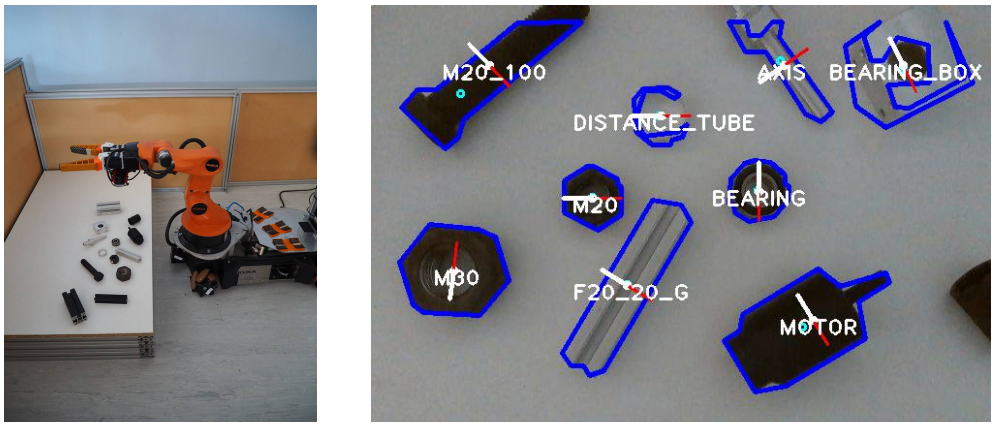
FIGURE 6.4: Additional RoCKIn@Work Manipulation objects: (a) Motor, (b) Distance Tube, (c) Axis, (d) Bearing, (e) Bearing Box

distributed in the arena. We are using up to three youBots as presented in Section A.1. The youBot provides a holonomic drive for flexible navigation and a mobile manipulator for grasping the objects. In Figure 6.2a, the three youBots are shown in their starting positions in the depot.

The objects that we used in this research were taken from the RoboCup@Work 2016 competition. The set consists of the original RoboCup@Work objects as shown in Figure 6.3 and a selected set of RoCKIn@Work objects, depicted in Figure 6.4.

6.3 Required Components for a Real World Application

In this section, we will present the required components for a real world application. The robots must be able to navigate safely within the environment. For this, the robots need to know where they are. Additionally, to be able to interact with the environment the robots need to be able to detect and manipulate objects, we introduce a reliable object detection method and for manipulation, we show how the kinematics of the robot arm can be computed. Moreover, in order to apply the commissioning Spatial Task Allocation Problem (SPATAP), as introduced in the previous chapter, we show the needed adaptations to cope with the physical restrictions of the robots. Lastly, we describe how complex behaviour for the robot can be created using a behaviour engine and state machines.



(a) (b)

FIGURE 6.5: (a) Pre-grip scan position. (b) Detected objects, classification and grasp positions.

6.3.1 Localisation, Navigation and Collision Avoidance

One of the most crucial capabilities of an autonomous agent is to localise itself efficiently in a known environment. To achieve this, we use gmapping (Grisetti et al., 2007) to build a map of the arena beforehand. The map is shown in Figure 6.2b. After the map is recorded it can be used by Adaptive Monte Carlo Localization (AMCL) (Fox et al., 1999) for efficient global localisation as also explained in Section 2.3.1.

Another necessary capability of the robots is to navigate in the known environment without colliding with obstacles. The map that was created with gmapping is used for the basic global navigation. The global path is computed by an A* algorithm and is then executed using the $COCALU^{dwa}$ approach as presented in Chapter 3. This allows the robots to navigate freely in the environment.

6.3.2 Object Recognition

Besides being able to autonomously navigate, object detection and recognition is crucial to be able to interact with the environment, i.e. picking up objects and placing them in the correct target locations. We use the openCV-library[1] to detect the objects. Figure 6.5 shows the detection in a service area. As can be seen the camera is facing the platform in a top down position.

We use the down-facing Intel Realsense camera which provides us with an RGB colour image and a depth image. We calibrated the camera so that the images are *registered*, which means that outputs are aligned such that the points in the RGB image correspond to the points in the depth image. However, since the depth camera has a much larger field of view than the RGB camera, the depth image has to be cropped to fit to the field of view of the RGB image. This means, that the registered image field of view is equal to the narrower field of view. Thus, while we lose some image parts, we gain the possibility to use the full RGB and depth information of the remaining points.

[1] http://opencv.org

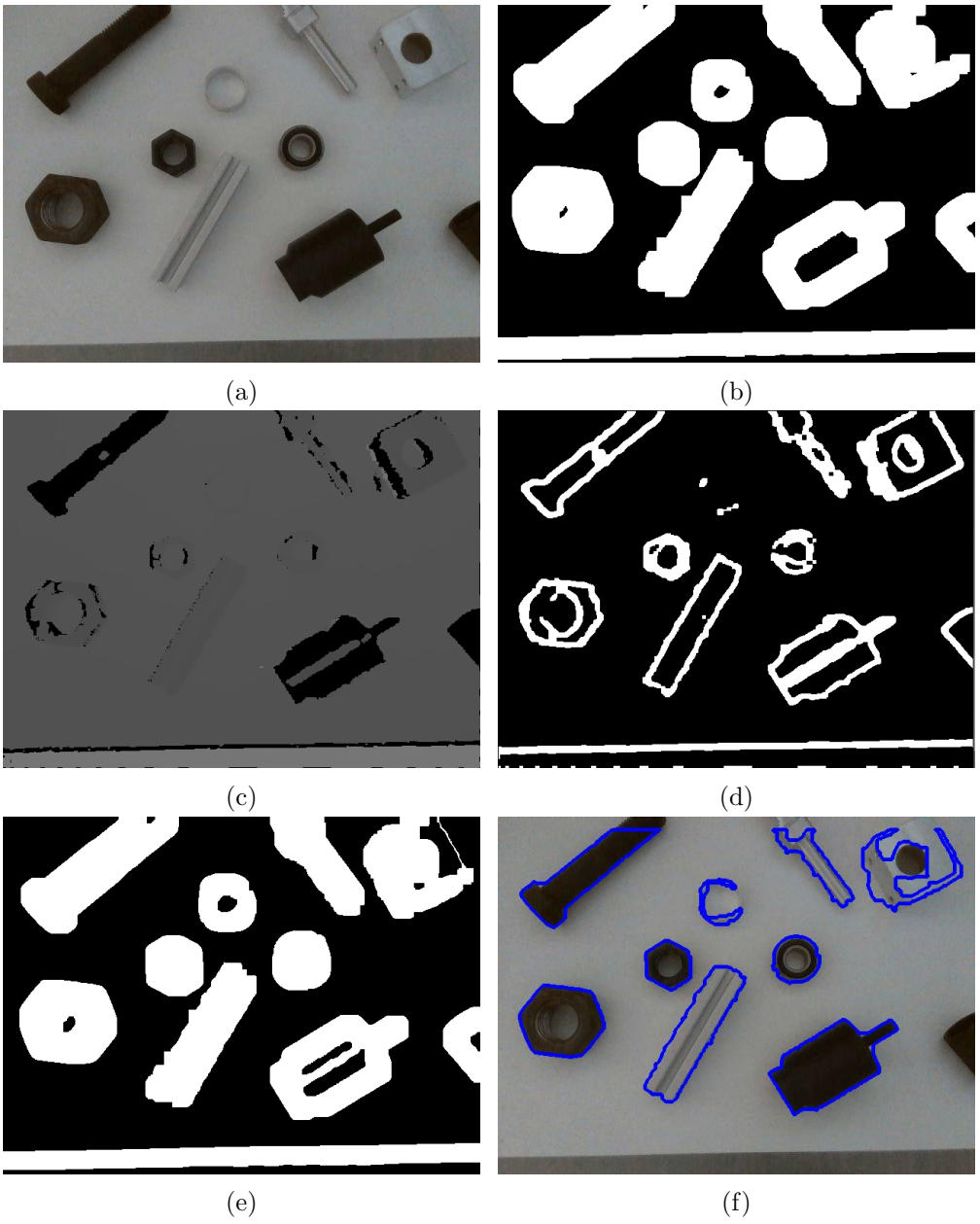


FIGURE 6.6: Object detection pipeline. (a) Input RGB image. (b) Processed RGB image. (c) Input depth image. (d) Processed depth image. (e) Combined images. (f) Detected contours of the objects.

This is necessary, since for instance, the depth camera uses an IR projection and detection to calculate the distances. Unfortunately, black objects are often not reflective enough for the IR camera, so that depth information is missing for those objects. Figure 6.6c shows the raw depth image. The shades of grey depict the measured distances, while a lighter colour represents larger distances. Black pixels mean that there is no depth information at that position. However, for the image processing, this can be used as a feature.

When looking at the colour image (Figure 6.6a), silver objects are highly reflective and in strong light these can appear on the RGB image to be as white as the background.

So they are difficult to detect purely based on the RGB image.

By processing the two images, RGB and image depth, separately and then using the combined contours, we were able to detect the objects much more reliably. Also, using the depth information, it is possible to use the *intrinsic camera parameters* together with the depth values to project the 2D pixel values from the image plane to a 3D world frame. This is necessary to be able to detect the objects at varying heights, since objects closer to the camera appear larger on the image that objects further away. In our setup, the camera is down facing, so that the image is a top-down view of the manipulation platform. Thus, the height can be calculated in a straight forward manner, since the camera height is known, subtracting the average depth of the points gives us the approximate platform height.

Figure 6.6 shows the object detection pipeline. For the processing of the RGB and depth image, we applied an adaptive threshold filter to the input images and the image is converted to black and white (see Figure 6.6 a-d). The white parts of the images are eroded and diluted multiple times as a methods to reduce the noise in the images. The two resulting processed images are combined (cf. Figure 6.6e) and this is used to detect the contours of the objects as shown in Figure 6.6f. After the contours are detected we can compute various features of the detected objects. We computed the following features:

1. Lengths of the principal axes and aspect ratio

   We create a rotated bounding rectangle around the contours, and the principal axes are then defined by the length and the width of this bounding rectangle. We use the lengths of these two axes to calculate the aspect ratio, by dividing the larger axis by the shorter axis.

2. Perimeter length, area of the contour and circularity

   We use the outline of the contour to calculate the length of the perimeter and the area of the contour. These two values can also be used to compute the circularity measurement:

   $$\frac{4 * \pi * \text{area}}{\text{perimeter}^2},$$

   This circularity measurement returns 1 for a circle and a value ≤ 1 for any other shape.

3. Mean intensity

   For all the points within the contour, we calculate the mean intensity value in the greyscale image.

4. Convex hull points and convexity defects

   We compute the convex hull of the contour and as one feature we count the number of points on the convex hull outline. Additionally, we count the number of convexity defects, which is the number of concave areas under the convex hull.

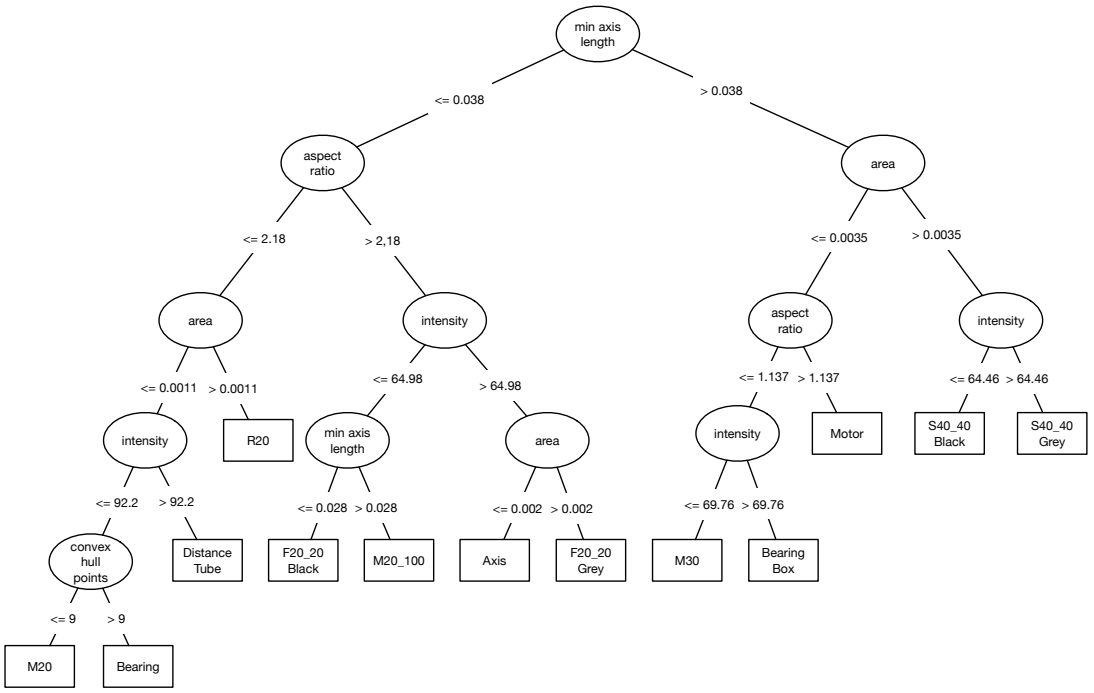


FIGURE 6.7: Learned J48 Decision Tree in WEKA.

We created a labelled data-set to train a J48 decision tree in WEKA (Hall et al., 2009) for the recognition of the objects. The J48 decision tree was chosen, since it provides human readable output, which is easy to interpret. Also the decision tree classification is a direct lookup, so it can be run in real time.

To create the training set, we placed each of the items on a rotating platform and recorded images from different positions of the camera and different rotations of the object. Afterwards, the features of the detected contours were extracted and this dataset was used to train the decision tree.

The resulting tree is shown in Figure 6.7. As we can see the features that were eventually necessary to distinguish the objects are min axis length, aspect ratio, area, intensity and the convex hull points.

Since we did not record *negative samples*, i.e. objects that do not belong to any of the classes, the decision tree will *always* return a valid class for every contour which is given to the decision tree. This means that the contour detection has to be tuned to exclude any contours that does not fit our objects. For this, we use the recorded dataset to determine the minimum and maximum values for each feature that was needed for the classification and contours which fall outside these values are excluded.

6.3.3 Inverse Kinematics for the youBot Arm

In order to manipulate the detected objects, the various joints of the arm have to be controlled such that the objects are grasped correctly. We implemented a simple inverse kinematics (McCarthy, 1990) module to calculate the joint values for any top-down

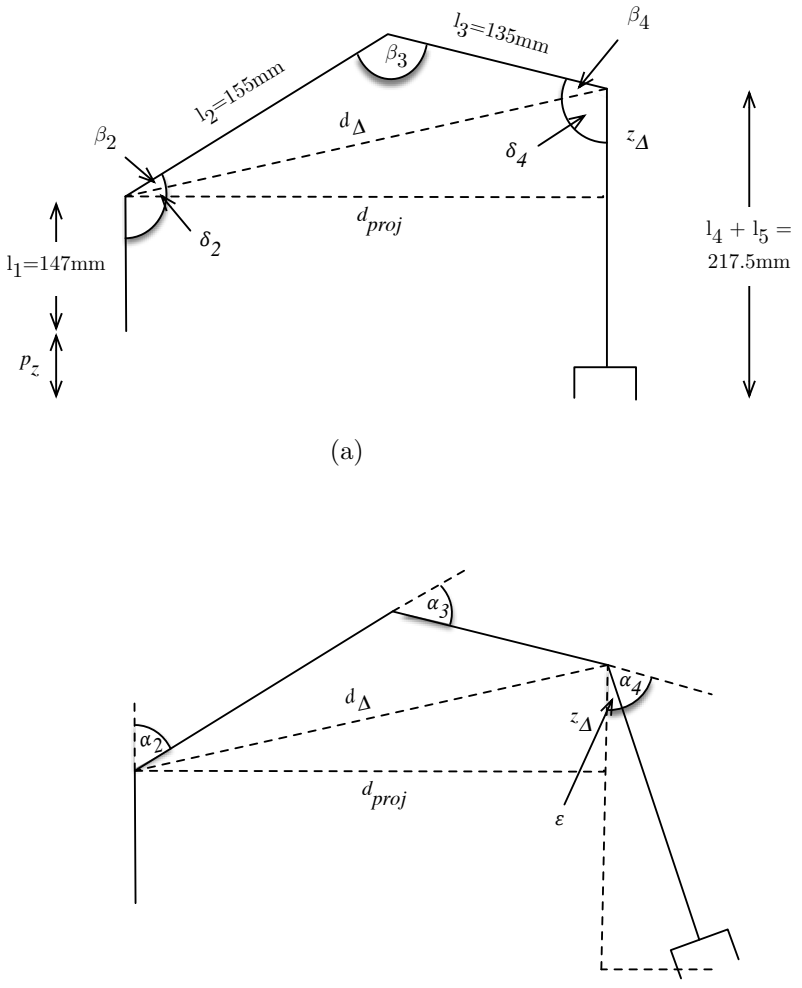


FIGURE 6.8: Simple inverse kinematics: By always gripping from a top down position shown in (a) or a slightly angled grasp (b), we can determine all angles for the joints.

gripping point that is in the reach of the robot. Since we are gripping from a top-down position, the inverse kinematics can be solved exactly, when we fix the first joint such that it is always pointing in the direction of the gripping point as shown in Figure 6.8a. Then the remaining joints can be calculated in a straightforward manner, by solving the angles of a triangle with three known side lengths, since we know the distance of the grip and also the lengths of all the arm-segments. We can also allow grasps in a certain angle ε, as shown in Figure 6.8b. As can be seen, one part of the end-effector now extends below the actual height. However, since we replaced the stock-endeffector with flexible fingers (cf. Section A.1), these are flexible enough to compensate for the height mismatch.

More specifically, the inverse kinematics can the be calculated as follows for any given point $p = (p_x, p_y, p_z)$ (given in the frame of the robot arm) with a grasp-rotation

(a) (b) (c)

FIGURE 6.9: Picking the F20 Black aluminium profile from the ground. (a) The object can be reached top down. (b) The object can be reached with some angle in the last arm segment. (c) The object is at the limit of the arms reach.

γ and end-link angle offset of ε as follows. The rotation angle for the first and last joint is calculated by:

$$\alpha_1 = -\operatorname{atan2}(p_y, p_x)$$
$$\alpha_5 = -\alpha_1 + \gamma$$

For the other joints, we need to calculate the projection distance of the two arm segments l_2 and l_3:

$$d_{proj} = \sqrt{p_x^2 + p_y^2} - \sin(\varepsilon)(l_4 + l_5)$$

Then, we need the difference in height of the grasp point p_z with respect to the length of the last joint including the endeffector length:

$$z_\Delta = p_z - l_1 + \cos(\varepsilon)(l_4 + l_5)$$

This allows us to calculate the length of the hypotenuse of the triangle l_2, l_3, d_Δ:

$$d_\Delta = \sqrt{d_{proj}^2 + z_\Delta^2}$$

With the lengths of the two joints l_2, l_3 and the hypotenuse d_Δ, we can calculate the angles:

$$\beta_4 = \arccos \frac{l_3^2 * d_\Delta^2 - l_2^2}{2 * l_3 * d_\Delta}$$
$$\beta_2 = \arccos \frac{l_2^2 * d_\Delta^2 - l_3^2}{2 * l_3 * d_\Delta}$$
$$\beta_3 = \pi - \beta_2 - \beta_4$$

We need to compensate for the tilt in the triangle:

$$\delta_2 = \arcsin \frac{z_{diff}}{d_\Delta}$$
$$\delta_4 = \frac{\pi}{2} - \delta_2 + \varepsilon$$

To achieve a top-down position the configuration is then defined as follows:

$$\alpha_2 = \frac{\pi}{2} - \delta_2 - \beta_2$$
$$\alpha_3 = \pi - \beta_3$$
$$\alpha_4 = \pi - \delta_4 - \beta_4$$

where $l_{x\in\{1,\dots,5\}}$ define the length and $\alpha_{x\in\{1,\dots,5\}}$ define the joint configuration of the arm segments, assuming that the zero position of the arm is pointing upwards for all joints.

The position-reproducibility of the arm is in sub-millimetre order, which means that if the same joint configuration is commanded twice, the final positions are within a millimetre difference. This proved to be sufficient for performing highly accurate grasp and place trajectories.

When combining the object recognition and the inverse kinematics of the arm, we can perform picks as shown in Figure 6.9. These picks show also the need to be able to grasp from a slightly angled positions, since as shown in Figure 6.9b and Figure 6.9c, the youBot would otherwise not be able to reach the object. In both pictures, the angle for the second link from the arm α_2, is already at its limit and the distance of the objects is larger the maximal distance possibly achieved by a top down grip $l_2 + l_3$. Thus, the end-link angle ε has to be used to be able to reach further. Since the inverse kinematics can be solved geometrically, we perform a loop with increasing ε until a valid solution is found or a maximum threshold for ε is exceeded. In the latter case the grasp fails and the robot has to recover by re-positioning itself closer to the object.

6.3.4 Commissioning Spatial Task Allocation Problems in the Real World

We use a *commissioning SPATAP* as defined in Definition 5.1. In order to be able to apply this approach, we created a navigation graph, on which the robots perform the planning. The graph is constructed by using the workstation locations and connecting the nodes to the closest other platforms. The graph is shown in Figure 6.10.

The actual navigation does not follow the edges in a strict manner. The navigation algorithm gets the target node as a goal and plans a path according to the currently observed obstacles and the other robots. Using the $COCALU^{dwa}$ method allows that the robots can move freely in the environment.

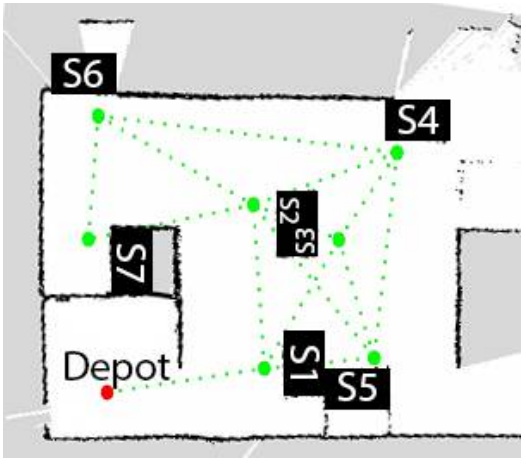


FIGURE 6.10: Annotated map with service platforms and navigation graph. The edges show only connectivity and not the suggested paths for the robots.

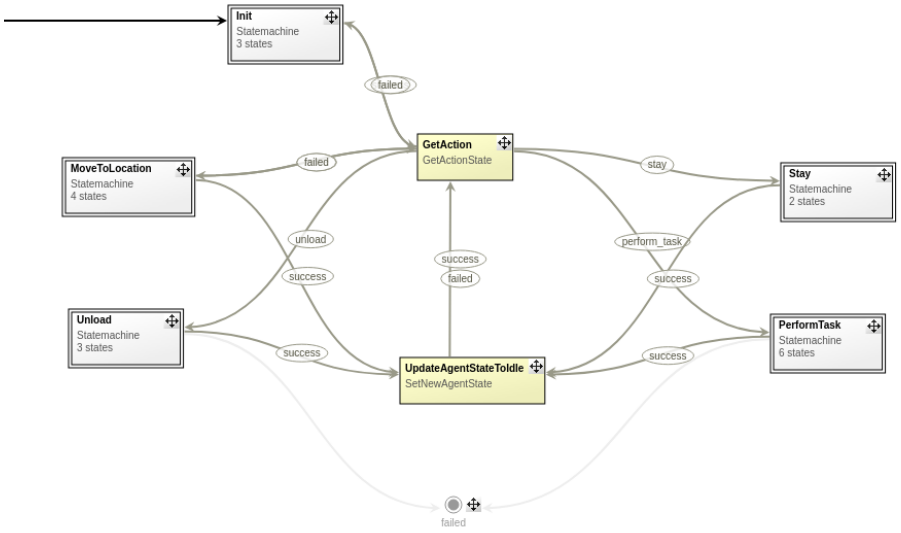
Additionally, there is a physical restriction that only one robot can perform a task at a node, since the platforms are not large enough to enable multiple robots to perform tasks at the same time. The navigation approach already prevents that the robots will collide, but in order to prevent inefficiencies, the robots continuously re-plan as soon as they are close to their target positions. In the following we will explain the robots' behaviour in more detail.

6.3.5 Robot Behaviour Creation

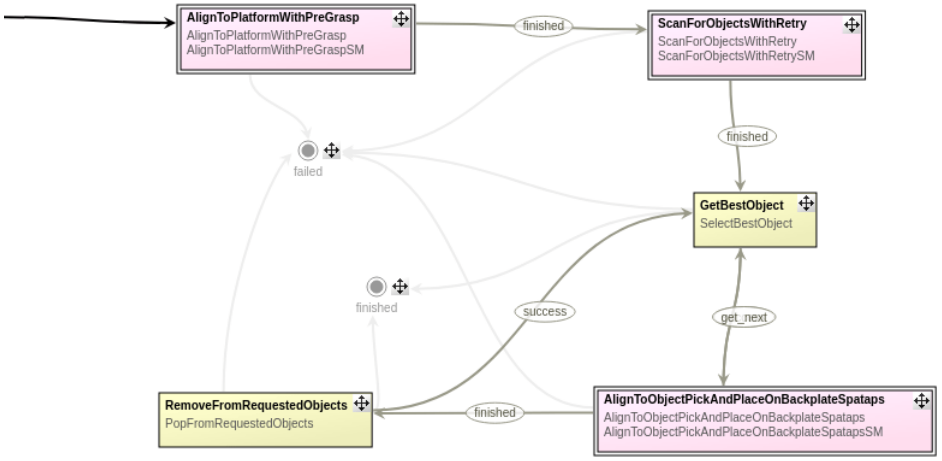
A robot acting in the real world has to be able to perform various complex tasks and behaviours. These behaviours can be captured state machines. For this, we use the flexible behaviour engine (FlexBe) [2]. It allows to create state machines and provides tools for consistency checking, some verification and visual inspection. It is developed for Robot Operating System (ROS). The tool has been deployed by the team ViGIR (Schillinger et al., 2016) in the DARPA robotics challenge (Pratt and Manzo, 2013) in which a humanoid robot had to perform various different complex tasks. These tasks were ranging from driving a vehicle, opening a door, walking through a debris field and operating a valve. Thus, it is well suited to capture complex robotic behaviour. It allows the creation of hierarchical state machines. We can capture sub-tasks, for instance recognising and localising an object or moving the robot arm, in a smaller state machine. These sub-task state machines can then be used to create more complex behaviours such as picking up an object at a service platform.

Figure 6.11 shows an example of the used state machines. States are shown in yellow and sub-task state machines in grey and pink. The most high level state machine is shown in Figure 6.11a. After a initialisation phase, in which the robot waits for the initial state, i.e. the currently active tasks and the position, the robot loops between the `GetAction` state and its different outcomes. During the `GetAction`, the planning

[2] https://flexbe.github.io



(a) Global state machine.



(b) Perform Task state machine.

FIGURE 6.11: FlexBe interface for creating state machines. States are shown in yellow and sub state machines are shown in grey and pink. (a) Global state machine. (b) PerformTask sub-task state machine.

algorithm is called, for instance the $MCTS^{it}_{Greedy}$, which looks up the current global state and calculates the best action. The outcomes are the possible actions in the commissioning SPATAP, i.e. `Stay`, `MoveToLocation`, `Unload` and `PerformTask`. These are then sub-tasks, which are handled in various sub-task state machines, such as the `PerformTask` state machine shown in Figure 6.11b. In the commissioning SPATAP the action `PerformTask` is corresponding to picking up an item at a location. Thus the robot has to align itself to the platform, scan for the object and grasp it.

Within this framework it is possible to capture failures, such that if a (sub-)task fails, it can be recovered. In our case the sub-task state machines always try to recover to the previous state of the robot. More specifically, the robot ensures that there is no item in the gripper and the arm is in a safe position for driving. Therefore, after a failure, the

TABLE 6.1: The confusion matrix for the object detection. The top row shows the detected labels and the first column shows the actual labels. The values are the percentage of the classification outcomes.

	F20 Black	F20 Grey	S40 Black	S40 Grey	M20 Bolt	M20 Nut	M30 Nut	R20	Dist. Tube	Motor	Bear.	Bear. Box	Axis
F20 Black	0.996	0	0	0	0.004	0	0	0	0	0	0	0	0
F20 Grey	0	1.000	0	0	0	0	0	0	0	0	0	0	0
S40 Black	0	0	1.000	0	0	0	0	0	0	0	0	0	0
S40 Grey	0	0	0	0.991	0	0	0	0	0	0.009	0	0	0
M20 Bolt	0.021	0	0	0	0.979	0	0	0	0	0	0	0	0
M20 Nut	0	0	0	0	0	0.966	0	0	0	0	0.034	0	0
M30 Nut	0	0	0	0	0	0.002	0.998	0	0	0	0	0	0
R20	0	0	0	0	0	0.006	0	0.994	0	0	0	0	0
Dist. Tube	0	0	0	0	0	0.007	0	0	0.992	0	0.001	0	0
Motor	0	0	0.001	0	0.005	0	0.013	0	0	0.981	0	0	0
Bear.	0	0	0	0	0	0.051	0	0	0.010	0	0.939	0	0
Bear. Box	0	0	0	0	0	0.002	0	0	0	0.004	0	0.994	0
Axis	0	0	0	0	0	0	0	0	0	0	0	0	1.000

robot loops back to the `GetAction` state and computes its new action.

We implemented `Unload` and `PerformTask` as *uninterruptable*, since during these actions the robot moves its arm and manipulates objects. This means that these state machines are executed until they succeeded or failed. On the other hand, during the `MoveToLocation` state machine, the `GetAction` remains active and computes new actions as soon as a new global state is observed and the robot is close to its target location. If the new action is also a `MoveToLocation`, the current navigation goal is updated to the target location. This ensures that the robots move smoothly between the various nodes even when the target node is occupied by another robot.

6.4 Experiments and Results

In this section, we present the experiments and results. In order to evaluate our object recognition and grasping performance, we perform various experiments. For the object recognition, we run 10-fold cross validation on the learned decision tree with the dataset that contains about 1000 instances of each object. For the grasping evaluation, we perform 10 picks of every item in different heights ranging from 0cm up to 20cm in 5cm steps.

Additionally, we show the results of the presented algorithms in a small scale real world warehouse commissioning setting. We use up to three youBots to simultaneously fetch items from the environment as presented in Section 6.2.

6.4.1 Object Recognition and Grasping

Table 6.1 shows the confusion of the object recognition approach when running 10 fold cross validation on the recorded dataset. We can see that there are very little mis-classifications. The most problematic classes are the *M20 Nut* and the *bearing*. These two items are very similar in size, shape and mean intensity value (cf. Figure 6.3 and Figure 6.4 for the objects). In Figure 6.5b and Figure 6.6a, the two objects are shown close to the centre of the image. As we can see from the learned decision tree (cf. Figure 6.7), the decision is than made on the number of points on the convex hull. Thus if the corners of the M20 Nut are not detected well, or the outline of bearing is detected with less corners, the object will be wrongly classified.

Also the F20 Black aluminium profile and the M20 Bolt were sometimes confused. As we can see from the tree, the dividing decision is made from the minimal axis length, which is for the bolt a bit larger, since it encompasses the head of the screw. Sometimes, the outlines are cropped since they are in the corners of the detection image. For instance, in Figure 6.6f, the M20 Bolt is slightly cropped at the top. If it would now be rotated by 180 degrees, the detection would probably fail.

Similarly, the Motor and the M30 Nut are confused. If the thin top part of the Motor is not detected or cropped, the aspect ratio changes, such that the classification will be an M30 Nut.

While this approach has be shown to be highly accurate and well suited for our needs, these examples show the shortcomings of this approach. Mis-classifications can still occur due to the previously mentioned reasons. The most common reason was that the contours were not correctly detected in the first place, such that either only a part of the item was detected or multiple items have been merged to a single contour in cluttered situations. In these cases the features are not corresponding to the learned classes.

There are many other options and possibilities to improve the detection and recognition method, for instance using the 3D point clouds instead of the separate RGB and depth images, but these exceed the scope of this thesis.

Besides the accuracy of the object recognition, we evaluate the grasping performance. We place each item in front of the robot at different heights at 0cm (on the ground), and on various tables of 5cm, 10cm, 15cm and 20cm height. The objects were manually placed in 10 positions with different rotations which were randomly generated. The youBot does not know these positions beforehand, thus it needs to look for the object, detect the position and height of the object, and calculate the inverse kinematics to perform the grasp.

As an example, Figure 6.9 shows three picks of the F20 Black aluminium profile from the ground and Figure 6.12 shows three picks each from the different platform heights. In Figure 6.13, we can see sample picks of the grasps of the various objects at a platform height of 15cm.

(a) (b) (c) (d)

FIGURE 6.12: Picks of the F20 Black aluminium profile from various heights. (a) 5cm. (b) 10cm. (c) 15cm. (d) 20cm.

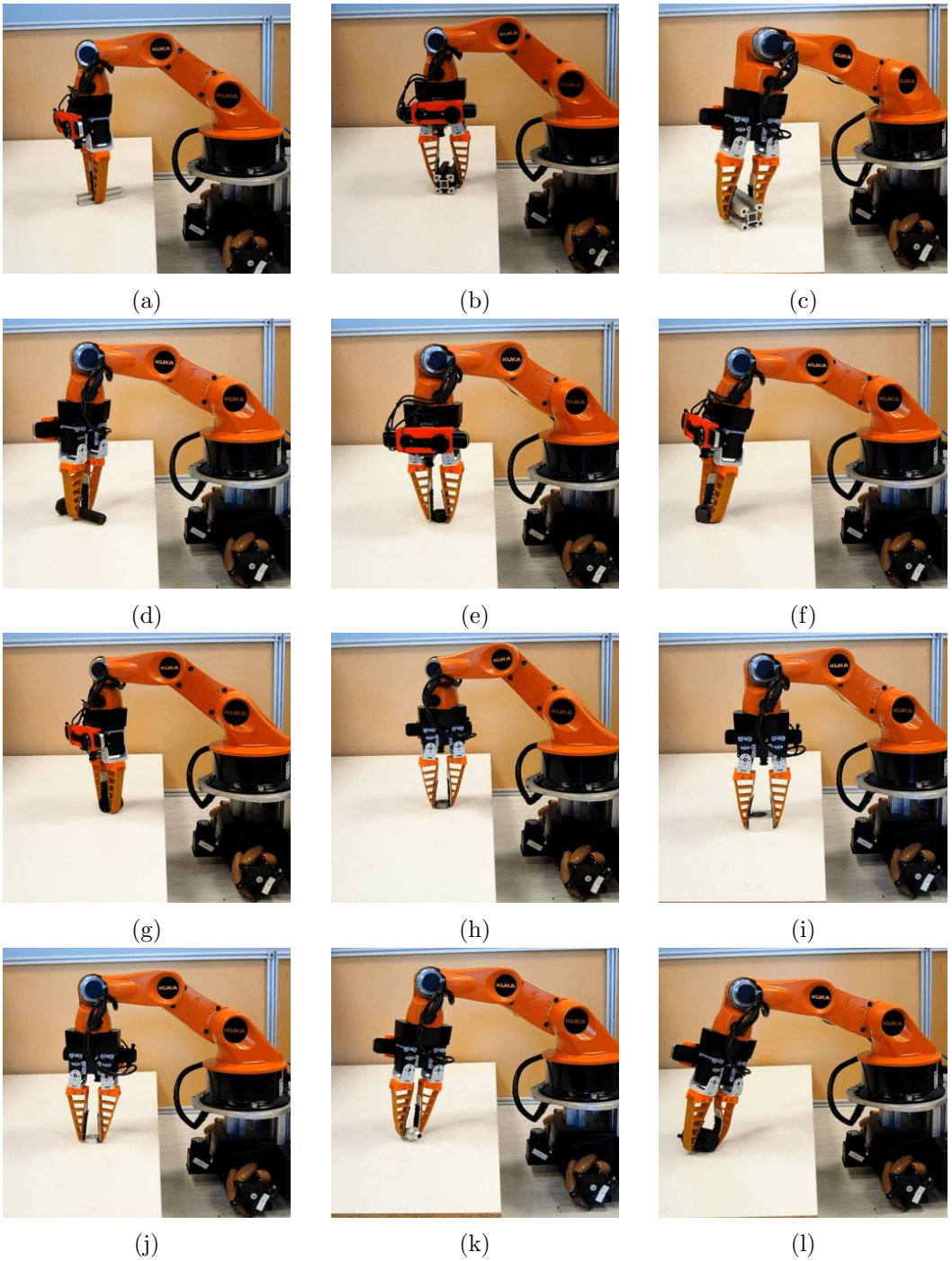
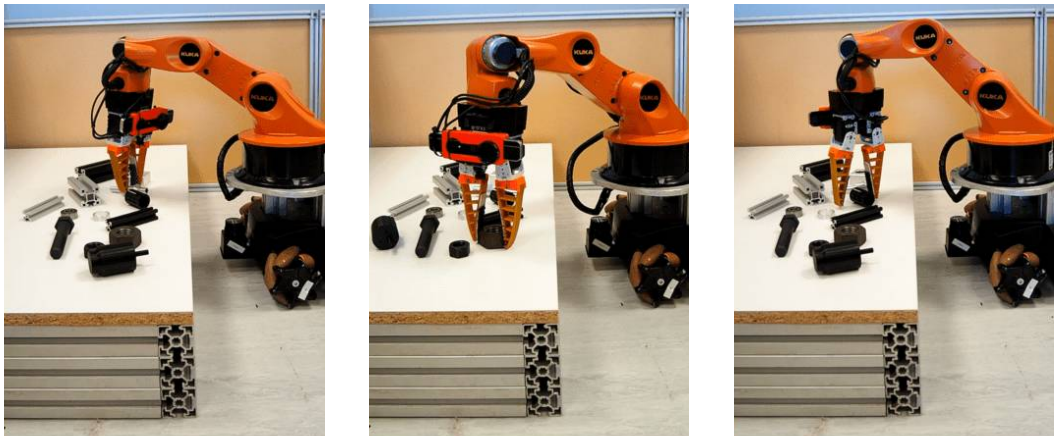


FIGURE 6.13: Picking the different objects from a platform at 15cm height. (a) F20 Grey. (b) S40 Black. (c) S40 Grey. (d) M20 Bolt. (e) M20 Nut. (f) M30 Nut. (g) R20. (h) Bearing. (i) Bearing Box. (j) Distance Tube. (k) Axis. (l) Motor

TABLE 6.2: Number of successful grasps for the various objects from different heights out of 10 trials.

	0 cm	5 cm	10 cm	15 cm	20 cm
F20 Black	9	10	10	10	10
F20 Grey	8	9	10	10	9
S40 Black	10	10	10	10	10
S40 Grey	10	10	10	10	10
M20 Bolt	9	10	10	10	9
M20 Nut	10	9	10	10	8
M30 Nut	10	10	10	10	10
R20	10	10	10	10	10
Dist. Tube	7	8	10	9	9
Motor	10	10	10	10	10
Bearing	9	10	10	10	10
Bear. Box	10	10	10	10	10
Axis	10	9	10	10	10



(a) (b) (c)

FIGURE 6.14: Picking objects in the presence of multiple other distracting objects. (a) Axis. (b) M30 Nut. (c) R30.

Table 6.2 summarises the results. It shows the number of successful grasps out of the 10 trials for each object at the different heights. As we can see, in most cases the grasps succeed. Especially the larger objects such as the Motor and the S40 Black and Grey are picked every single time. On the other hand the Distance Tube proved the most difficult to grasp. This is due to the small size and low height of the object, thus the detection has to be very precise. Also if the grasp has to be at an angle, since a top down grasp is out of reach, it can happen that the gripper cannot get low enough to grasp the object. For instance, if we look at Figure 6.13l and Figure 6.13c, we can see the effect of an angled grasp with the compliant fingers of the gripper. The finger which is further away from the arm is hovering slightly above the platform height, while the other finger gets deformed quite a bit. This effect is more pronounced on larger items, since the fingers move at an arc. This is in contrast to smaller items, where the gripper

TABLE 6.3: Generated orders.

Time	0s	40s	50s	90s	110s	120s	130s	150s	160s	190s
Location	S4	S5	S4	S4	S3	S7	S2	S7	S5	S7
Priority	5	1	1	5	1	2	5	1	1	1

can close the fingers further as shown in Figure 6.13k.

When comparing the picks at different heights (Figure 6.12), we can see that for 5cm and 10cm the second and third arm segment from almost a parallel line to the ground. Thus, when revisiting Figure 6.8, we can see that $d_{proj} \approx l_2 + l_3$, meaning that in these heights the arm can reach the furthest, while for 15cm and especially 20cm we can see that the two arm segments point upwards, an the projected distance will be more limited, i.e. $d_{proj} < l_2 + l_3$. We would expect that the performance is also the best in these heights, which can be confirmed, as for 10cm every grasp succeeded for every object.

We also tested the approach for picks in more cluttered environments as shown in Figure 6.14. In these cases, we placed all items on the table and requested a grasp of a specific item. The robot was able to pick all the items.

To summarise, the inverse kinematics model together with the shown object detection method, we are able to perform grasps for all the objects on various different heights.

6.4.2 Warehouse Commissioning Results

After evaluating that the main components are working to our satisfaction, for the main evaluation, we implemented a small scale warehouse environment, in which orders are simulated to come in over time, and these need to be picked by the robots and brought back to the depot.

For the model of how the orders can appear, we use a discrete *step* which takes 10 seconds. After each step, new orders can appear at every pickup location with probability $p = 0.03$.

The orders appear in three different priorities values, $P_{low} = 1$, $P_{medium} = 2$ and $P_{high} = 5$. The probability distribution for the different priorities is $p(P_{low}) = 0.8$, $p(P_{medium}) = 0.1$ and $p(P_{high}) = 0.1$. Thus generally, the tasks appear with low priority, while sometimes medium or high priority orders appear.

We run the setup for three and a half minutes, i.e. 210 seconds, in which the orders appeared according to the model, and afterwards we stop new orders from appearing and let the robots run until the last order is picked up and returned to the depot. The orders that were generated are shown in Table 6.3. For the locations refer to Figure 6.2.

We evaluate this setting with two and three youBots which are all equipped with the sensors and modifications as explained in Section A.1. The youBots run all the code onboard and are connected to a shared WiFi over which they broadcast their own positions and receive broadcasts from the warehouse management system, which

periodically sends out the current state of the system with 5Hz. The youBots use the $COCALU^{dwa}$ implementation as introduced in Chapter 3 for navigation.

We use the commissioning SPATAP, with the adaptations as explained in Section 6.3.4. We compare the $Greedy^{it}$ algorithm with $MCTS^{it}_{Greedy}$ as introduced in Chapter 5 for deciding which action to take next. We repeat each approach 10 times for two and three youBots.

As evaluation, we calculate the weighted average active time (WAAT) for an order, i.e. the time in seconds from the appearance (t_a) of the order until being picked by a robot(t_{end}), weighted with the priority:

$$WAAT = \sum_{t \in \mathfrak{T}} P_t * (t_{end} - t_a), \tag{6.1}$$

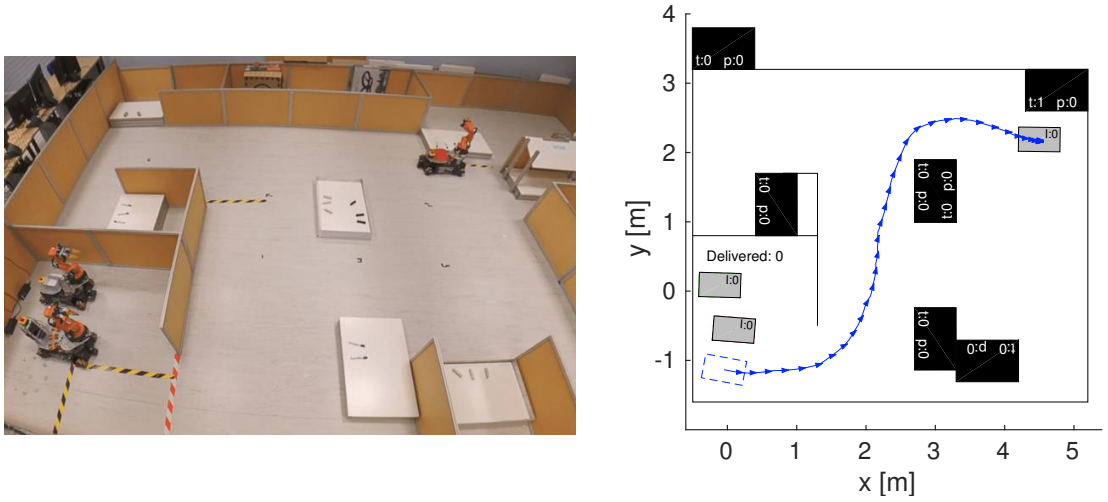
where $\mathfrak{T}$ are the appeared tasks over the whole run and P_t is the priority of the task. If a task is not picked yet at an evaluation moment, the current time is chosen as t_{end}. Additionally, we evaluate the overall finishing time, i.e. when the last robot has returned to the depot and has unloaded, and record the average driven distance, and average linear and angular jerk per robot.

Figure 6.16 shows an overview of a run using the $MCTS^{it}_{Greedy}$ approach and Figure 6.15 shows a run $Greedy^{it}$ each with 3 youBots. The pictures of the real world state are shown on the left, and a graphical representation of the state is shown on the right. It shows the currently active tasks on the platforms (a), the already picked tasks (p) and the inventories of the robots (I). Also the number of tasks delivered back to the depot is shown.

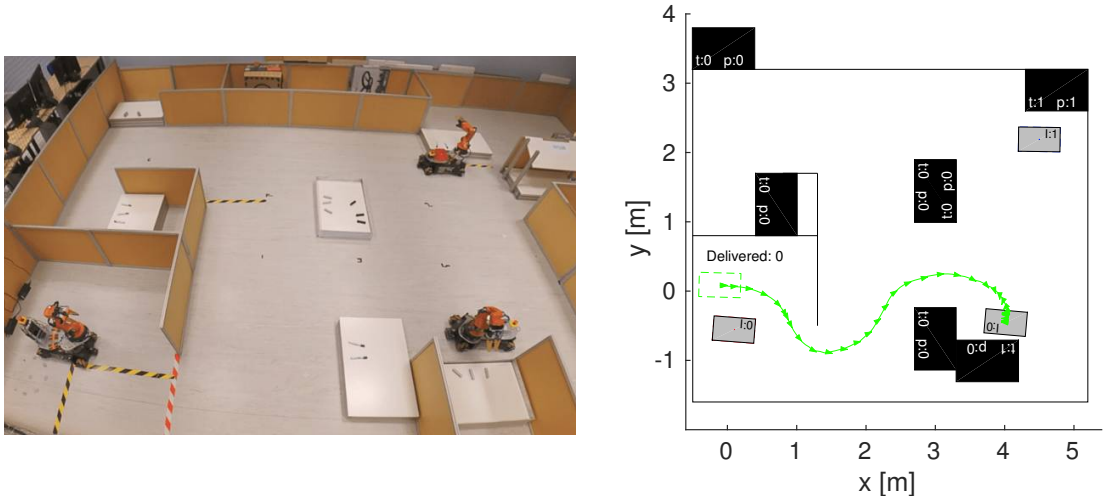
Something that is directly visibly when comparing the two figures is that with using the MCTS approach, the robots drive more, even though no tasks are present. After 30 seconds, all three robots are dispersed, while with $Greedy^{it}$ only one robot is active after 30 seconds, and it takes more than 120 seconds until the last robot is assigned a task.

This is due to the nature of the simulated appearance of the tasks as shown in Table 6.3. At first only an item from top right corner (S4) is ordered. After 30 seconds another task becomes active at the lower right platform (S5), then another two tasks appear at S4, and only after 110 seconds an item is ordered at a third platform (S3) and after 120 seconds from S7.

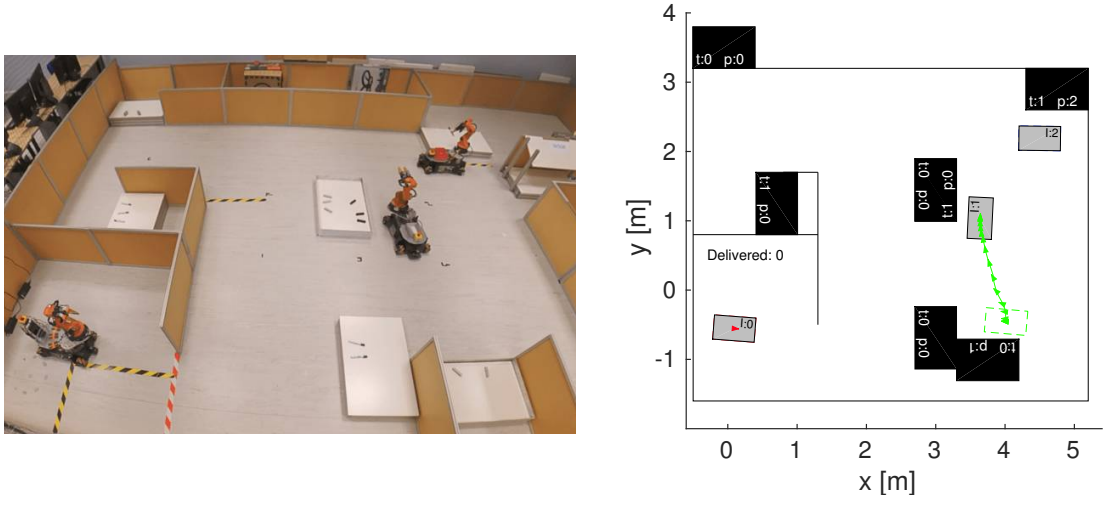
The $Greedy^{it}$ algorithm, shown in Figure 6.15, performs as expected. The youBot with highest ID (blue traces) gets the first task assigned at S4, since the other youBots run exactly the same algorithm, they reach the same conclusion and they remain stationary. The next ranked youBot (green traces) performs the tasks at S5 and is finished quickly enough to perform the task that appears at S3. The highest ID youBot is still busy at the top right corner, since new tasks have appeared during the pickup. Thus only after the task appears at S7, the last youBot (red traces) comes into play since at that point both other youBots are still busy performing tasks. At that time the highest



(a) 30s

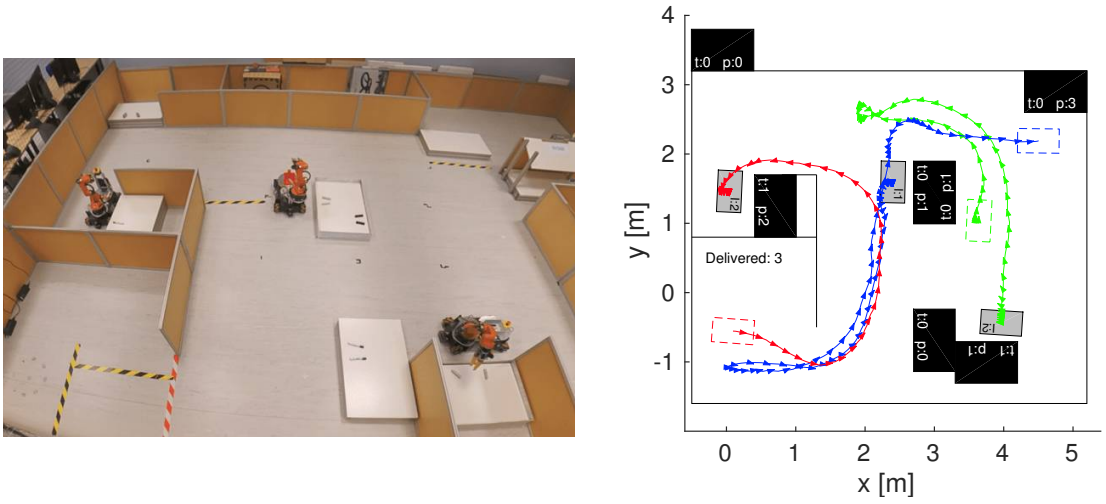


(b) 60s

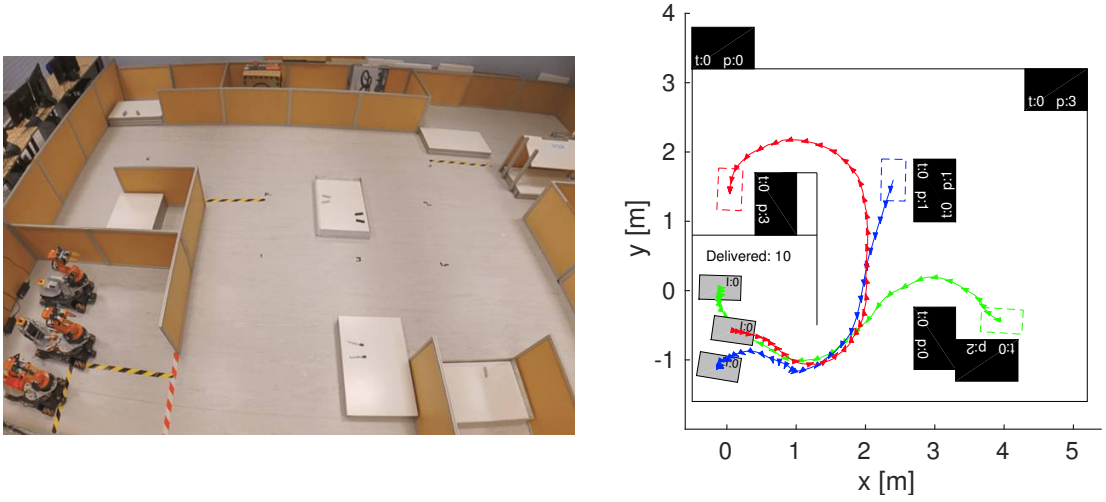


(c) 120s

FIGURE 6.15: Real world run with *Greedy*^{it} and three youBots. Showing the picture of the overhead camera (left) and the state (right), consisting of the driving traces, active tasks (t) and picked tasks (p) on the workstations and the inventories of the robots (I). (a) after 30s. (b) after 60s. (c) after 120s.



(d) 210s

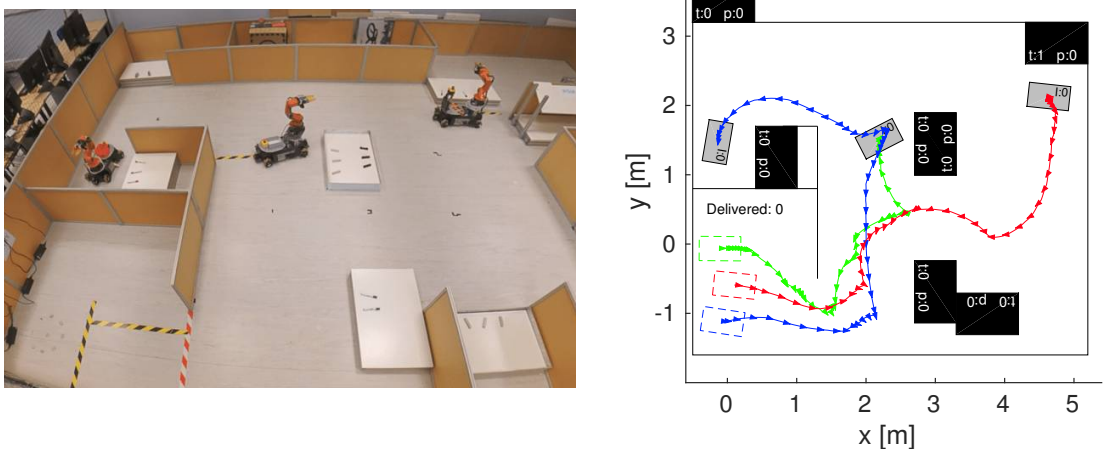


(e) end, 275s

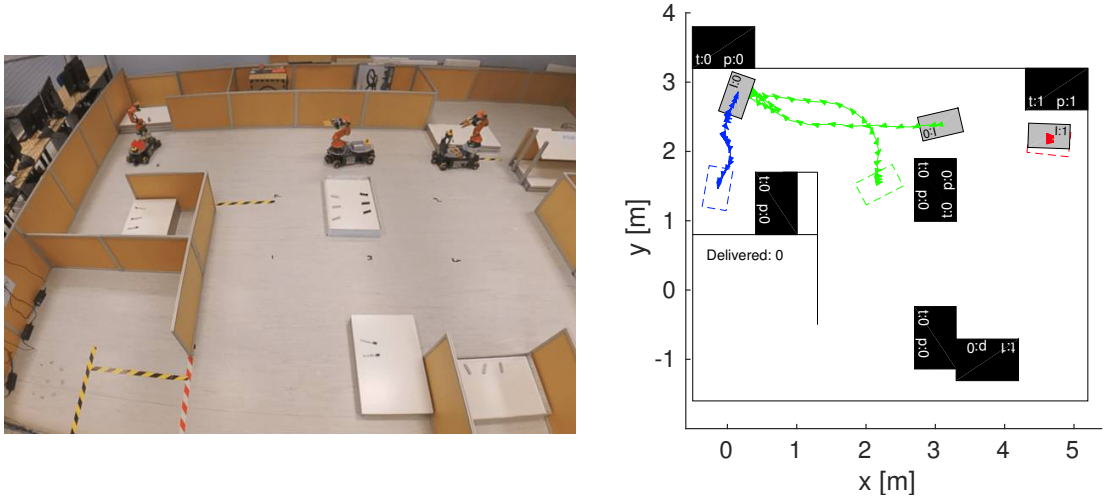
FIGURE 6.15: Real world run with $Greedy^{it}$ and three youBots continued. Showing the picture of the overhead camera (left) and the state (right), consisting of the driving traces, active tasks (t) and picked tasks (p) on the workstations and the inventories of the robots (I). (d) after 210s. (e) at the end, after 275s.

ID youBot has picked up three items, thus it has to drive to the depot to unload before performing another task. After 210 seconds only three tasks remain to be picked and at each location already a robot has arrived. Thus, the tasks are picked and the robots drive back to the depot.

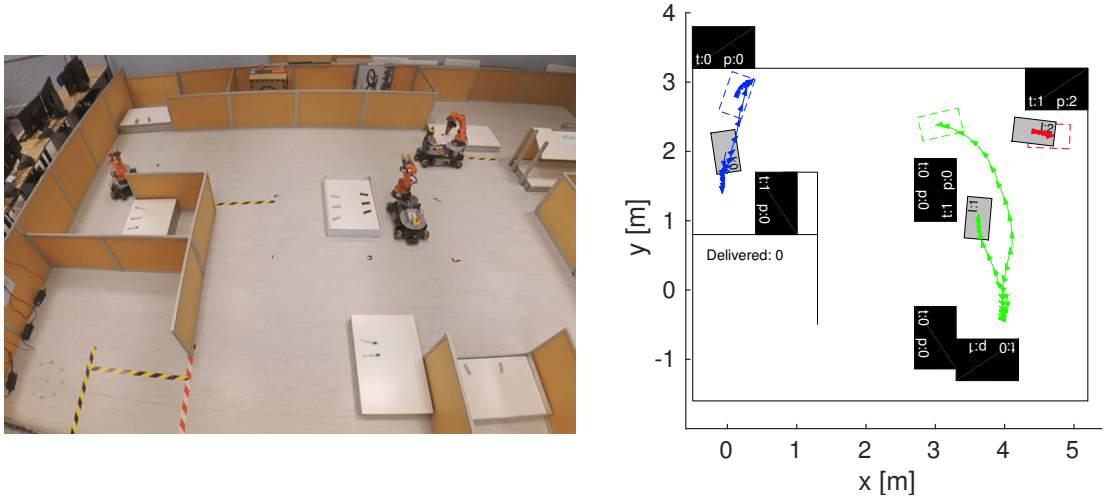
When inspecting the $MCTS^{it}_{Greedy}$ run, shown in Figure 6.16, we can see that already in the first 30 seconds all robots are moving out and distributing themselves in the environment. After 60 seconds, when comparing the state with the $Greedy^{it}$ approach, we can see that this time the distribution actually was at a disadvantage, since the robot is a bit further away from the task. But after 120 seconds, it worked as an advantage, since a tasks appeared at S7, and one youBot is in close vicinity of the platform, while with the other approach the robot has to drive from the depot all the way to the task. In the end, we can see that the finishing times are almost identical for



(a) 30s

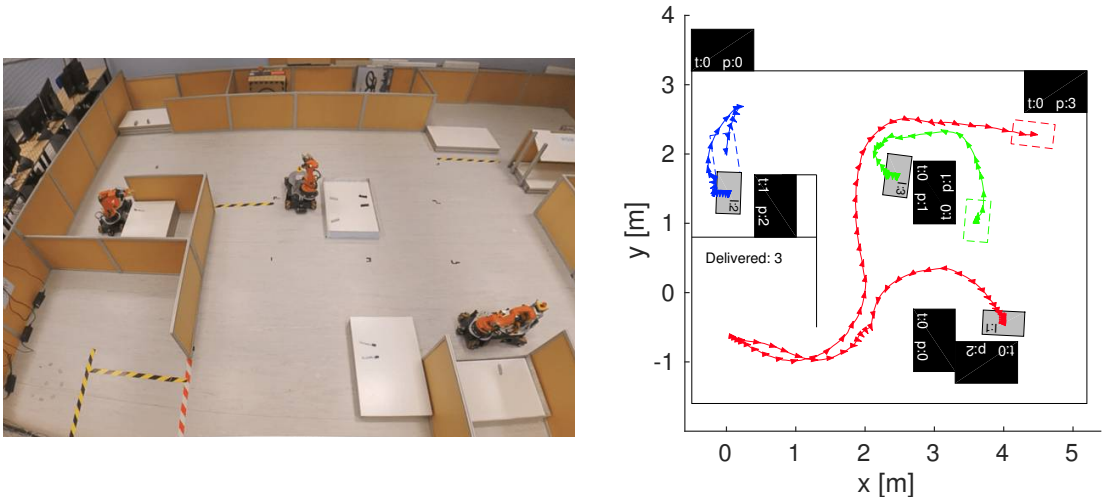


(b) 60s

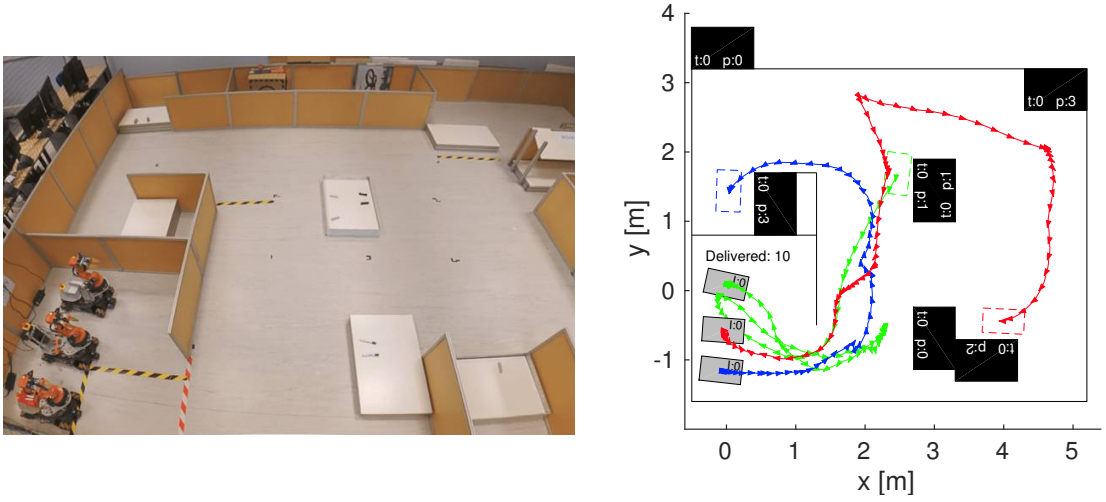


(c) 120s

FIGURE 6.16: Real world run with $MCTS^{it}_{Greedy}$ and three youBots. Showing the picture of the overhead camera (left) and the state (right), consisting of the driving traces, active tasks (t) and picked tasks (p) on the workstations and the inventories of the robots (I). (a) after 30s. (b) after 60s. (c) after 120s.



(d) 210s

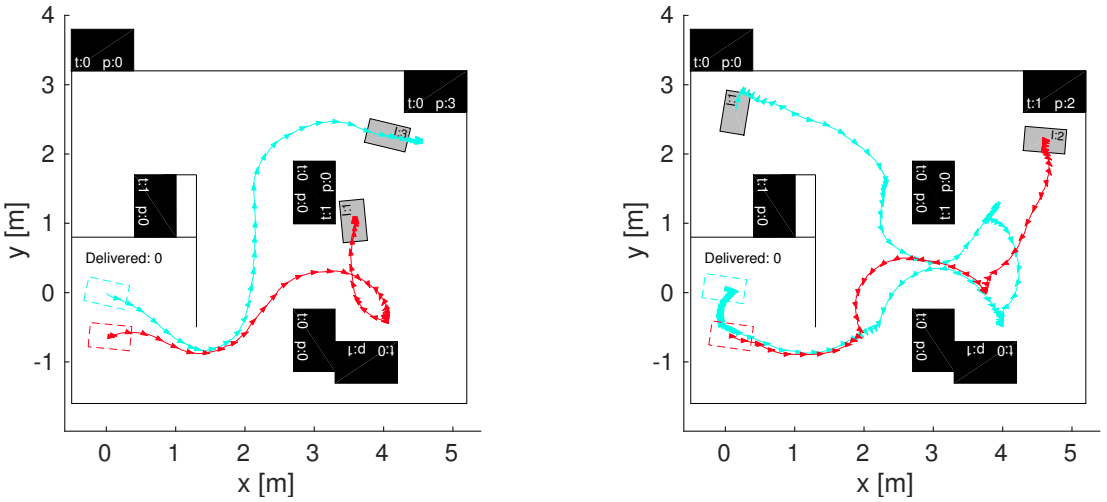


(e) end, after 286s

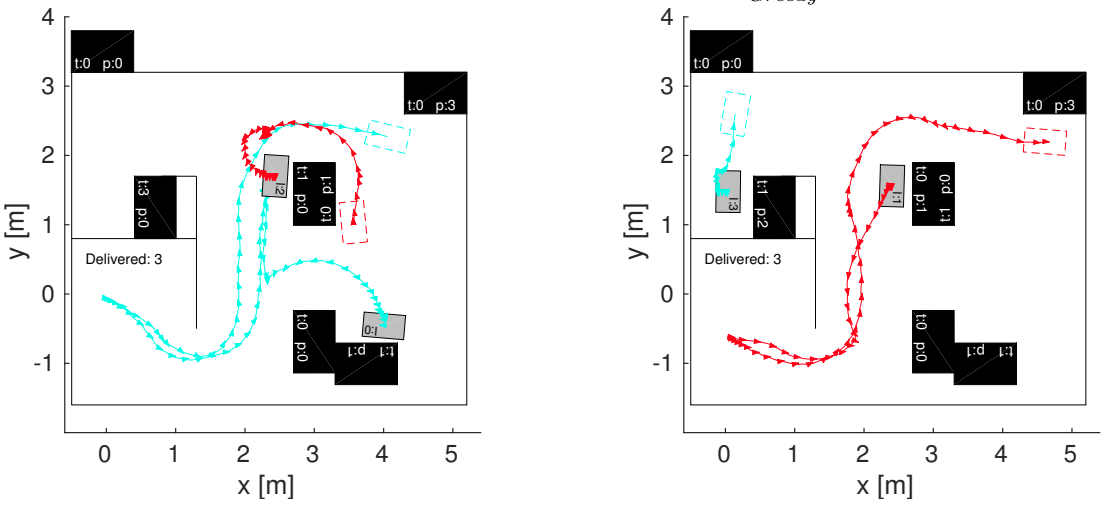
FIGURE 6.16: Real world run with $MCTS^{it}_{Greedy}$ and three youBots continued. Showing the picture of the overhead camera (left) and the state (right), consisting of the driving traces, active tasks (t) and picked tasks (p) on the workstations and the inventories of the robots (I). (d) after 210s. (e) at the end, after 286s.

the two approaches, while the $MCTS^{it}_{Greedy}$ take slightly longer to finish. This is due to $MCTS^{it}_{Greedy}$ spreading out, thus the robots are not at the depot when all tasks are picked, thus all three robots have to drive back at the same time, while with $Greedy^{it}$, a robot that did not get a new task after unloading remains at the depot.

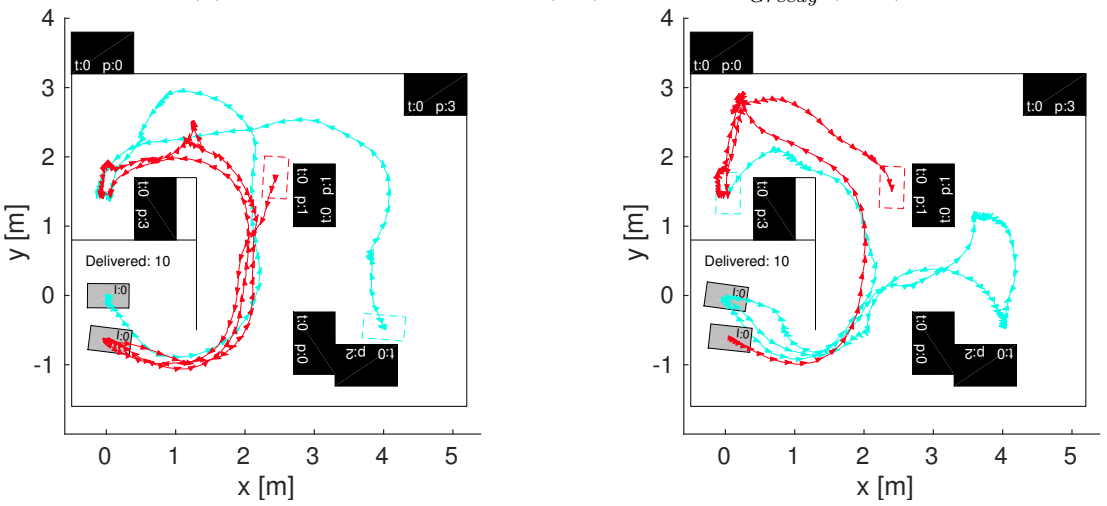
In Figure 6.17 a run with two youBots is compared. After 120 seconds (cf. Figure 6.17a), both approaches are similar far. $Greedy^{it}$ approach was a bit faster in the top right corner, thus the youBot is already finished and starting to drive back to the depot, while with $MCTS^{it}_{Greedy}$, both robots moved out at the same time. This resulted in more complex paths. After 210 seconds, shown in Figure 6.17b, there are still three tasks active at S7 (left) with $Greedy^{it}$, and another one at S2 (centre) and S5 (lower right) each, while $MCTS^{it}_{Greedy}$ has only one task left at S7, and the same amount of tasks at S2 and S5. Thus again, the positioning of $MCTS^{it}_{Greedy}$ enabled it to get to the



(a) State after 120s, $Greedy^{it}$ (left) and $MCTS^{it}_{Greedy}$ (right)

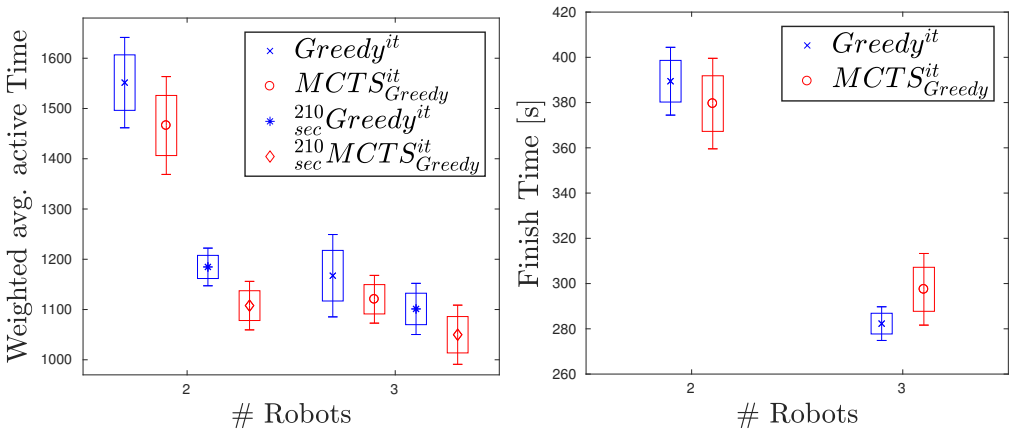


(b) State after 210s, $Greedy^{it}$ (left) and $MCTS^{it}_{Greedy}$ (right)



(c) end, $Greedy^{it}$ (left) after 402s, $MCTS^{it}_{Greedy}$ (right) after 383s

FIGURE 6.17: Real world run comparing $Greedy^{it}$ (left column) and $MCTS^{it}_{Greedy}$ (right column) and two youBots. Showing the state, consisting of the driving traces, active tasks (t) and picked tasks (p) on the workstations and the inventories of the robots (I). (a) After 120s, (b) after 210s and (c) at the end, after 402s for $Greedy^{it}$ and 383s for $MCTS^{it}_{Greedy}$.



(a) Weighted average active Time.

(b) Finish Time.

FIGURE 6.18: Evaluation of the two approaches. (a) The weighted average time the tasks were active is shown. Evaluation after 210 seconds and at the end. (b) The overall finishing time.

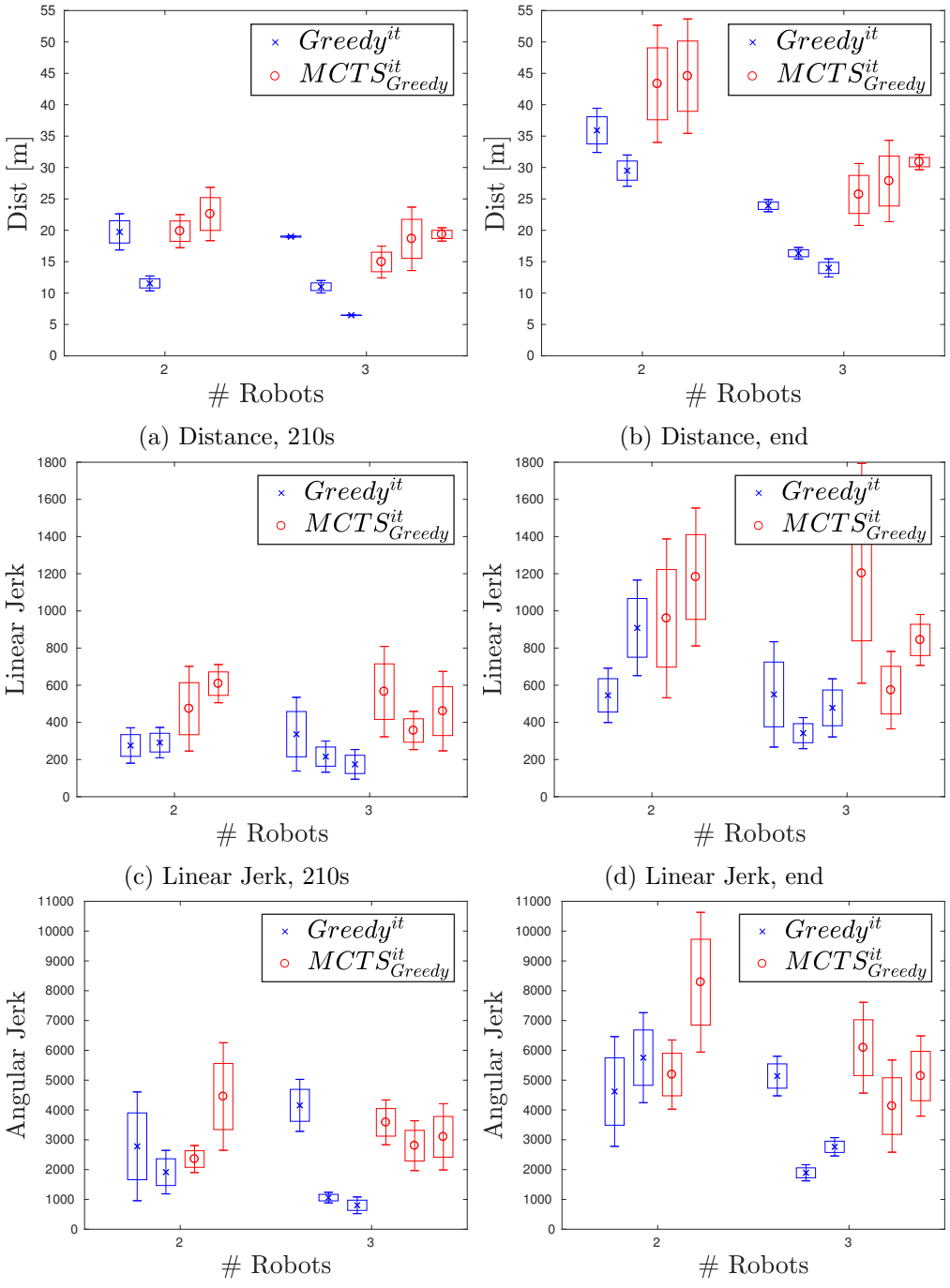
tasks at S7 much quicker than with $Greedy^{it}$. Which is also reflected in the finishing times.

The quantitative results for exhibit a similar result. Figure 6.18 shows the results for the finishing time and WAAT. Examining Figure 6.18a, we can see that the WAAT is slightly better when using the $MCTS^{it}_{Greedy}$ approach. However, this result is only significant after 210 seconds with two robots. When comparing the results at the end of the run, they are very close together. After 210 seconds, the stopped the appearance of the tasks according to the model. Nevertheless, $MCTS^{it}_{Greedy}$ still plans as if the tasks will appear. Therefore, it continues to position itself, it might not take the direct and shortest routes to the remaining tasks. On the other hand, when running the $Greedy^{it}$ approach one youBot will always drive the shortest way an active task.

For three robots, both approaches yield very similar performance, with a slight, but not significant advantage for the $MCTS^{it}_{Greedy}$ approach. As we can also see from Figure 6.15, showing the $Greedy^{it}$ run with three robots, the third robot is not needed until 120 seconds, only then all three youBots are performing tasks simultaneously. Thus, there is not much advantage when running $MCTS^{it}_{Greedy}$ in comparison to $Greedy^{it}$.

When looking at the differences in WAAT after 210 seconds, after which no new tasks appeared, and the finish time, we can see that there is much larger difference with two robots than with three robots. This is explained, when comparing the finish times Figure 6.18b for the approaches. With three youBots, the approaches can keep up reasonably with the appearing tasks, such that after 210 seconds almost all tasks are already finished, while with two youBots, a lot of tasks still remain to be done.

Interestingly, we can see that $MCTS^{it}_{Greedy}$ actually finished later with three robots than $Greedy^{it}$, while with two robots both results are almost on-par. This is as explained before, as $MCTS^{it}_{Greedy}$ spreads out, when the last pick is finished and the robots are called back to the depot, they are usually further away than $Greedy^{it}$. With two youBots, both robots are active until the end, thus the difference is not as visible.



(a) Distance, 210s (b) Distance, end

(c) Linear Jerk, 210s (d) Linear Jerk, end

(e) Angular Jerk, 210s (f) Angular Jerk, end

FIGURE 6.19: Evaluation of 10 runs with two and three youBots running $Greedy^{it}$ and $MCTS^{it}_{Greedy}$. Each youBot is evaluated independently, shown in decreasing ID from left to right. The whiskers show the standard deviation, and the boxes the 90% confidence interval. Evaluation after 210 seconds (a, c, e), and after the run was finished (b, d, f) for distance, and linear and angular jerk.

When comparing the distances driven and jerk results for the robots, we can see that the advantage of $MCTS^{it}_{Greedy}$ is achieved with significantly higher driving distances and jerk costs. Figure 6.19 shows the results per robot with decreasing ID from left to right.

As $Greedy^{it}$ is deterministic, all robots receive the same tasks in the same situation, therefore the deviation of the results is usually much smaller than with $MCTS^{it}_{Greedy}$. Nevertheless, since the robots are driving using free navigation, and the robots continuously re-planned while driving, it happens that the different runs yield different behaviour. We can see that the, higher ID robots have also a significantly higher driving distance than with $MCTS^{it}_{Greedy}$, where the distances are more evenly spread. This is because $MCTS^{it}_{Greedy}$ is already probabilistic in its nature due to the MCTS rollouts. Therefore, the robots do not act the same in any two runs.

An advantage of the decentralised approach is that if a robot brakes down, the algorithm reacts accordingly. Since the robot does not broadcast its position anymore, the other robots will not plan with that robot, i.e. they will not predict that the broken robot will do any tasks and therefore, the approach reacts online to the break down of a robot.

Similarly, if a new robot is added to the system, the other robots automatically react to that as soon as they receive the position of that robot. This approach assumes that the other robots are actually collaborating to a common goal and actually perform *sensible* behaviour and do not try to mislead the each other by broadcasting wrong information. As a result, the approach in the current form is aimed at collaborative settings were a team of robots has a common goal and the robots have no incentive to perform more or less tasks than the other robots.

To conclude, the results show that running an approach like $MCTS^{it}_{Greedy}$ can be advantageous over a more simple heuristic like $Greedy^{it}$, especially is settings in which the coordination is crucial. This is for instance in cases where the number of tasks is about equal to or just above the number of robots, as in the setting with two youBots. However, it can also be seen that this advantage is achieved with the trade-off of much higher driving distances.

6.5 Summary

In this chapter, we have shown how the navigation approach COCALU as presented in Chapter 3 can be combined with the SPATAP framework from Chapter 4 and Chapter 5 to implement a real world warehouse commissioning application. We have introduced and evaluated the necessary additional components, i.e. object recognition and inverse kinematics of the robot arm, and we have present the $MCTS^{it}_{Greedy}$ and $Greedy^{it}$ approach running on two and three robots. While $MCTS^{it}_{Greedy}$ performs better in terms of weighted average active times of the tasks, this comes at a costs of higher driven distances and the resulting higher costs in linear and angular jerk. Additionally, when we stop the appearance of tasks, the performance of $MCTS^{it}_{Greedy}$ decreases. Hence, as soon

as the model used for planning does not correspond to the actual model anymore, this should be taken into account. Nevertheless, we show that the robots are able to perform this complex task without communicating the concrete actions between the robots and each robot running the approach onboard. The action selection is solely based on the commissioning SPATAP as presented in Chapter 5. More specifically, only the global state is broadcast by the warehouse management software. Actually, if the robots could *see* or *detect* the tasks and other robots by other means, e.g. by some form of radar or using cameras, no communication would be needed at all, which is in contrast to the usual approaches that need more sophisticated communication methods.

Chapter 7

Conclusions

To conclude this thesis, we will review the main contributions and limitations of this work and discuss possible directions for future work.

7.1 Contributions

In this work, we have studied decentralised solutions for Multi-Robot Systems (MRS), with the focus on applications in the physical world. We conducted the research based on the research questions introduced in Chapter 1. In the following, we aim to answer these questions.

To what extent can we ensure decentralised collision free navigation in a highly dynamic setting considering that other robots, humans and other known and unknown obstacles might be present?

In Chapter 3, we have shown how we can combine an onboard localisation method, namely Adaptive Monte Carlo Localization (AMCL), with the Velocity Obstacle (VO) paradigm to enable safe navigation of multiple robots in a common workspace. We have shown how to take the robots' own localisation uncertainty into account and how the system can be implemented only based on local communication, sharing the robots positions and using a common reference frame. These prerequisites are in many cases easily satisfied, especially in industrial environments where even global communication might be possible through WiFi or other systems.

We have introduced a sampling based approach, which enables better control of the robots' behaviours using cost functions. Combining this approach with a human detection method allows for safe navigation in the presence of other robots and humans.

While this approach was already able to avoid dynamic and static obstacles, it treated static obstacles simply as stationary robots leading to problems in tight situations. As a solution, we have combined the sampling based approach which the Dynamic Window Approach (DWA). As a result, it is possible to use the forward simulations of the DWA

to avoid obstacles in configuration space and we can avoid dynamic obstacles, such as other robots and humans using the translation in velocity space.

In summary, by combining sensor-based localisation with local communication we are able to allow free navigation in highly dynamic settings in which other robots, humans and static obstacles are present.

To what extent can we derive a formal framework within which Multi-Robot coordination can be described, understood and evaluated?

To answer this research question, in Chapter 4, we revisited the current research in Multi-Agent Systems (MAS), and more specifically the Multi-Agent Markov Decision Process (MMDP) framework. We defined a sub-class of problems within this framework, which we called Spatial Task Allocation Problems (SPATAPs).

While there are algorithms that converge to optimal policies for solving MMDPs, such as *Policy Iteration* and *Value Iteration*, these algorithms do not scale anywhere near to the problem sizes that we are considering. To tackle this problem, we identified a number of *subjective approximations* that exploit the characteristics of the problem. While these approximations allow us to approximate the *aggregated effect* of the other agents, further approximations are necessary. We introduced *phase approximations*, that, combined with the previous approximations, reduce the state and actions space complexity to tractable sizes without any exponential dependence on the number of agents or the number of possible locations.

To further understand and evaluate the framework, we performed an empirical evaluation, comparing against a state of the art partitioning approach and where possible (for very small problem sizes) against the optimal solution. We show that our approach yields near optimal solutions while outperforming the partitioning approach.

To conclude, using the SPATAP framework, it is possible to define many problems as a subclass of MMDP. While for most problem sizes it is computationally intractable to solve them optimally, we can exploit the underlying structure and use approximations. Using this framework, we can formally analyse and evaluate the performance of various different algorithms.

To what extent can this formal framework be used to implement and deploy decentralised Multi-Robot Systems?

While we introduced in Chapter 4 a formal framework and a first algorithm to deal with various problems in the MRS domain, the framework needed to be extended to deal for instance with transportation tasks as commonly occurring in the warehouse commissioning domain, where the items have to be picked up and transported to a drop-off depot.

In Chapter 5, we define the *commissioning SPATAP*, which covers these warehouse commissioning problems. We impose maximum capacity constraints on the robots.

While the previous approach can still be applied to this problem, it did not scale well enough for large real world sized problems.

As a solution, we show how Monte Carlo Tree Search (MCTS) can be used in the SPATAP framework to alleviate the need for the previous approximations and to enable the required scalability to allow for online planning even in larger problem sizes. We introduce cheap rollout strategies that are inspired by Sequential Single Item (SSI) auctions and an online partitioning approach.

We empirically show that the MCTS approach leads to significantly higher performance especially in larger and more complex problems, when compared to our previous approach. Moreover, we show that the approach scales better, yielding high performance on a warehouse with over 200 locations and an 8-robot team.

To what extent can the developed framework and algorithms be deployed in an Industry 4.0 context, considering single robots, multiple robots and humans?

Since the previous results have been obtained in simulation, we present the implementation of a physical warehouse environment in Chapter 6 to further investigate the applicability in an Industry 4.0 context. Since we have shown in Chapter 3 that the navigation approach is able to deal with humans in the workspace, we have focused on evaluating the planning framework for a single and multiple robots.

We introduce the necessary additional components for the robots when acting in a physical environment. The MCTS approach introduced in Chapter 5 enables decentralised online planning on the robots. The physical runs show the advantages of a decentralised system. The robots can be removed and added on-the-fly without needing to reconfigure the system. We evaluate the differences between the greedy heuristics and the MCTS approach on up to three youBots and show that while the MCTS approach has better performance in terms of the weighted average active time of the orders, this comes at a trade-off of higher driving distances. Moreover, in the physical setting, the advantage of the MCTS was more apparent with two robots than with three robot. To summarise, the MCTS approach is more effective when the model is accurate and when there are more tasks than robots, since then the planning helps to coordinate more efficiently.

7.2 Applications to other domains

The application of the decision making approach is not limited to the specific warehouse commissioning task as shown in Chapter 6. Every time that a team of robots has to fulfil tasks that are at different locations, this idea can be applied. More specifically, as long as the environment can be formalised as SPATAP, this approach can be applied.

Let us assume for example an urban taxi environment, where the customers have to be picked up by a team of (autonomous) taxis, this would a similar application, since

we can also model where it is likely that a taxi is needed in the future. In such an environment, our approach would be well suited. Another different application, could be for instance in a search and rescue environment, where a lot of houses have to be searched and the robots have to decide to which house to go next.

Our approach plans with the current global state only, i.e. the positions of the robots and the locations for the tasks. This is the only communication required, and this information can be broadcast even with an unreliable communication method. Some parts of the state could even be observed by different means such as vision.

Other approaches, like auctioning, need to communicate in two ways, i.e. they need to react on the information they receive and reply, thus they need a reliable communication method and a protocol that they have to adhere to. If in our approach a message is not received, the robots will automatically react to that and plan accordingly, i.e. if a robot breaks down and does not broadcast it's position anymore, it will not be taken into account during the planning, and the tasks will still be carried out.

7.3 Limitations of the approach

The main limitation of the approach is the assumption that the robots have access to the complete state, i.e. in the warehouse commissioning example, all the currently active orders and all positions of the other robots. While this is usually feasible in industrial applications, in less controlled environments this assumption can sometimes be hard to fulfil, because communication might be limited. However, we have shown in Chapter 4 that even if the global state is approximated, the approach yields good results. This result is promising to alleviate this limitation. Additionally, an option would be to extend the framework towards Partially Observable Markov Decision Processes (POMDPs), which are able to handle with partial-observability at the cost of a more complex model.

The presented algorithms are developed for a collaborative setting. Thus, the approach assumes that the other robots are in fact collaborating towards a common goal and perform "sensible" behaviour, e.g. they do not try to mislead each other by broadcasting wrong information. For instance, if the robots are coming from different stakeholders, this might not necessarily be the case. To counter this limitation, the robots should have no incentive to perform more or less tasks than the other robots.

7.4 Future Work

In this thesis we have investigated the applications of decentralised Multi-Robot Systems in real world settings. In Chapter 3, we introduced a collision avoidance algorithm that is relying on local communication for robot-robot detection. This prerequisite can be alleviated in a straight forward fashion by implementing a vision based detection method as described in (Tuyls et al., 2016). In (Bareiss and Berg, 2015), the control obstacles were introduced, which maps the configuration space into the control space

of the robots. This enables a more general representation of the control inputs that will lead into collision. Using these control space obstacles instead of the obstacles in velocity space, can possibly lead to even better performance.

In Chapter 4, we have focused on deriving a formal model that we can use to describe and evaluate various problems in the MRS domain. While we are able to compare against an optimal solution, this is usually only feasible for very small problem sizes. Thus, an interesting direction for future work would be to investigate quality guarantees for the proposed approximations and approaches. Additionally, the notion of empathy relates to the field of plan and intent recognition. It would be interesting if the presented approximations are applicable in the area of research as well.

Another valuable benchmark could be made by comparing against repeatedly applying SCRAM (MacAlpine et al., 2014) and/or the Hungarian method (Kuhn, 1955) after each time step. This would allow for these approaches that the allocation might be changed during the execution. While these approaches are centralised, they could be run on decentralised every robot, since the result is deterministic.

Additionally, extending the model to allow for other interaction settings, such as positive interactions, i.e. tasks for which multiple robots are required. Other future work includes to investigate the effect, if only parts of the tasks would be known, and/or the information of the other robots is inaccurate, missing or delayed.

The work in Chapter 5 has focused on extending the SPATAP framework to a more complex application and introducing a scalable solution method that can run decentralised and online on the robots. Some possible routes for future work are for instance, tuning the node evaluation function `NV` (Equation 5.1). We can include a weighted connectivity of the nodes as a value. More specifically, we can compute a force-field, based on the task appearance probabilities and the locations of the agents. This could lead to even more improved positioning, especially when little to no tasks are active even for the non searching approaches. Other possibilities are to improve the MCTS search, e.g. by introducing node priors as shown in (Gelly and Silver, 2007) or using deep neural networks to approximate the value of complex states as has been done in AlphaGo (Silver et al., 2016).

In Chapter 6, we show a proof-of-concept of a real world warehouse commissioning system. This is achieved by the combination of the navigation approach as introduced in Chapter 3 and the decision making approach (Chapter 5). Interesting directions of future work would be to scale the system further, i.e. larger environments and more robots. Additionally, introducing the collaborations with humans would be an interesting field of future work.

Appendix A

Robot Systems

In the following the two robot platforms are introduced and the changes and modifications that are needed for our experiments will be highlighted.

A.1 youBot

The youBot is an omni-directional platform that has four mecanum (Hon, 1975) wheels. It comes with a 5-degree-of-freedom arm that is made from cast magnesium, and has a 2-degree-of-freedom gripper. The platform is manufactured by KUKA[1]. It has been designed to work in industrial like environments and to perform various industrial tasks. With this open-source robot, KUKA is targeting educational and research markets (Bischoff et al., 2011). Figure A.1a shows a model of the stock youBot.

The arm is 655 mm high, weighs 6.3 kg, and can handle a payload of up to 0.5 kg. The working envelope of the arm is 0.513 m^3, and it is connected over EtherCat (Jansen and Buttner, 2004) with the internal computer, and has a power consumption limit of 80 Watts. The position accuracy and repeatability of the arm is within 1 mm. The gripper has two detachable fingers that can be remounted in different configurations. The gripper has a stroke of 20 mm and a reach of 50 mm, it opens and closes with an approximate speed of 1 cm/s.

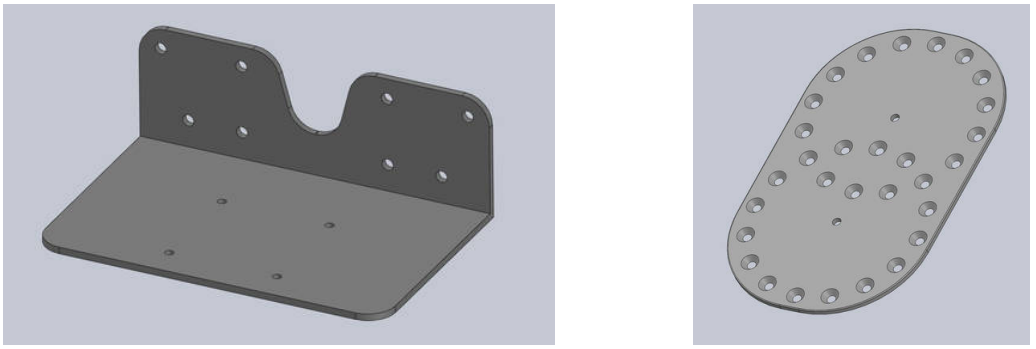
In order to meet the requirements we demand from the youBot platform, we made a number of modifications to the robot. In this paragraph we describe which parts are modified and why these modifications are a necessity for our approach. Figure A.1b shows the modified youBot setup. The major modification is the gripper, which is replaced by two FESTO FinGripper fingers mounted on two Dynamixel AX-12A servo motors. This increases the stroke to more than 20 cm and the speed of the gripper to up to 10 cm/s. Also the fingers passively adapt to the shape of the objects as shown in Figure A.3.

To extend the reach of the robot-arm in respect to the chassis, we designed an extension plate of 5 mm thick aluminum (see Figure A.2b). This plate can extend the arm towards the bounds of the chassis, and is designed to be a multi-purpose extension

[1] http://kuka.com

(a) (b)

FIGURE A.1: (a) A stock youBot. (b) smARTLab modified youBot.



(a) (b)

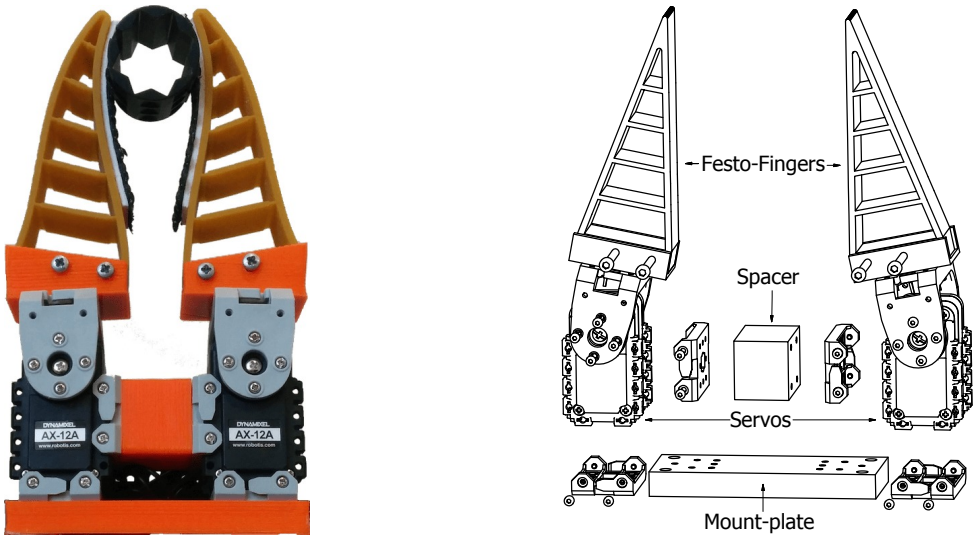
FIGURE A.2: (a) a CAD model of the laser mounts, (b) a CAD model of the arm extension plate.

for the youBot arm. Additionally, the position of the arm is elevated by 10.5 cm to compensate for the longer manipulator.

For perceiving the environment, two Hokuyo URG-04LX-UG01 Light Detection and Ranging (LIDAR) sensors are mounted parallel to the floor on the front and back of the robot. For mounting these LIDAR sensors a special laser mounting bracket was designed (see Figure A.2a). In contrast to the thin aluminium stock brackets, these custom brackets are constructed from 4mm thick steel in order to prevent deformation, reduce vibration and to ensure constant horizontal alignment.

In order to detect and recognise manipulation objects, an Intel Realsense SR300[2] RGBD camera is attached to the last arm joint. This camera is mounted so that it faces perpendicular away from the manipulator, as can be seen in Figure A.1b. This mounting technique allows for an unobstructed view of the manipulation platform. While this also means that the camera cannot be used for visual servoing, we found that the arm can be controlled precisely enough to make this disadvantage negligible.

[2] On some of the robots, the predecessor F200 is mounted.



(a) (b)

FIGURE A.3: (a) Illustration of the adaptation of the gripper to the round shape of the object. (b) CAD drawing of the separate parts of the new designed gripper in more detail.

The base computer is upgraded from the stock Intel Atom based architecture to an Intel i7-6700T CPU and is powered by a dedicated 4 cell 14.8V / 8000mAh Lithium Polymer battery pack that is charged and monitored by an OpenPSU power unit. For cooling we mounted an additional fan in the base of the robot.

For safety reasons the robot is equipped with an emergency stop button, that stops all robot movement, without affecting the processing units, so when the stop is released the robot can continue its movement without having to re-initialise it again.

(a) (b)

FIGURE A.4: (a) CAD model of a stock Turtlebot. (b) smARTLab modified Turtlebot.

A.2 Turtlebot

The Turtlebot[3] platform is a low-cost personal robot with limited resources. This robot is equipped with a laptop with core-i3 CPU for computation. We use the second generation, a Turtlebot II, for which a custom base was developed by Kobuki. As a main sensing unit the Turtlebot is originally equipped with a Microsoft Kinect RGBD sensor as shown in Figure A.4a, but in our setup a Hokuyo URG-04LX-UG01 LIDAR sensor has been added.

Figure A.4b shows the final configuration. The LIDAR is mounted on the top plate. For static obstacle detection, we use the information of the LIDAR sensor together with three bumpers that are located in the front half of the robot. Furthermore, the robot estimates its position by integrating the wheel odometry and gyro information together with the sensor readings.

[3] http://www.Turtlebot.com/

Abbreviations

General

FoF	Factory of the Future
LIDAR	Light Detection and Ranging
LLSF	Logistics League sponsored by Festo
MAS	Multi-Agent System
MRS	Multi-Robot System
ROS	Robot Operating System

Navigation

AMCL	Adaptive Monte Carlo Localization
CALU	Collision Avoidance under Localization Uncertainty
COCALU	Convex Outline Collision Avoidance under Localization Uncertainty
DWA	Dynamic Window Approach
ORCA	Optimal Reciprocal Collision Avoidance
SLAM	Simultaneous Localization and Mapping
VO	Velocity Obstacle
RVO	Reciprocal Velocity Obstacle
HRVO	Hybrid Reciprocal Velocity Obstacle

Decision Making

Dec-MDP	Decentralized Markov Decision Process
Dec-POMDP	Decentralized Partially Observable Markov Decision Process
MCTS	Monte Carlo Tree Search
MDP	Markov Decision Process
MMDP	Multi-Agent Markov Decision Process
MRTA	Multi-Robot Task Allocation
OAP	Optimal Assignment Problem
OSI	Ordered Single Item
POMDP	Partially Observable Markov Decision Process
PSI	Parallel Single Item
SPATAP	Spatial Task Allocation Problem
SSI	Sequential Single Item

Bibliography

Ahmed, A., Varakantham, P., and Cheng, S. "Uncertain Congestion Games with Assorted Human Agent Populations". In: *Proc. of the Conference on Uncertainty in Artificial Intelligence*. 2012, pp. 44–53.

Ahuja, R. K., Magnanti, T. L., and Orlin, J. B. "Network flows". In: (1988).

Alers, S., Tuyls, K., Ranjbar-Sahraei, B., Claes, D., and Weiss, G. "Insect-inspired robot coordination: foraging and coverage". In: *Artificial life*. 2014, pp. 761–768.

Alonso-Mora, J., Baker, S., and Rus, D. "Multi-robot navigation in formation via sequential convex programming". In: *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE. 2015, pp. 4634–4641.

Alonso-Mora, J., Breitenmoser, A., Rufli, M., Beardsley, P. A., and Siegwart, R. "Optimal Reciprocal Collision Avoidance for Multiple Non-Holonomic Robots". In: *Proc. of the Internationanl Symposium on Distributed Autonomous Robotic Systems DARS 2010, Lausanne, Switzerland, November 1-3, 2010*. 2010, pp. 203–216.

Alonso-Mora, J., Naegeli, T., Siegwart, R., and Beardsley, P. "Collision avoidance for aerial vehicles in multi-agent scenarios". In: *Autonomous Robots* 39.1 (2015), pp. 101–121.

Althoff, D., Kuffner, J., Wollherr, D., and Buss, M. "Safety assessment of robot trajectories for navigation in uncertain and dynamic environments". In: *Autonomous Robots* 32 (3 2012). 10.1007/s10514-011-9257-9, pp. 285–302.

Amador, S., Okamoto, S., and Zivan, R. "Dynamic Multi-Agent Task Allocation with Spatial and Temporal Constraints". In: *Proc. of the AAAI Conference on Artificial Intelligence*. 2014, pp. 1384–1390.

Amato, C., Konidaris, G., Anders, A., Cruz, G., How, J. P., and Kaelbling, L. P. "Policy search for multi-robot coordination under uncertainty". In: *The International Journal of Robotics Research* 35.14 (2016), pp. 1760–1778.

Amato, C., Konidaris, G., Cruz, G., Maynor, C. A., How, J. P., and Kaelbling, L. P. "Planning for decentralized control of multiple robots under uncertainty". In: *Proc. of the IEEE International Conference on Robotics and Automation*. IEEE. 2015, pp. 1241–1248.

Associated Press. *Robot kills man at Volkswagen plant in Germany*. Ed. by apnews.com. May 2015. URL: https://apnews.com/d18c4801a5324926a1845690148b664a.

Auer, P., Cesa-Bianchi, N., and Fischer, P. "Finite-time Analysis of the Multiarmed Bandit Problem". In: *Machine Learning* 47.2-3 (May 2002), pp. 235–256.

Bareiss, D. and Berg, J. van den. "Generalized reciprocal collision avoidance". In: *The International Journal of Robotics Research* 34.12 (2015), pp. 1501–1514.

Baye, M. R. and Beil, R. O. *Managerial economics and business strategy*. Vol. 5. McGraw-Hill New York, NY, 2006.

Becker, R., Zilberstein, S., Lesser, V., and Goldman, C. V. "Transition-independent decentralized Markov decision processes". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2003, pp. 41–48.

Beetz, M., Klank, U., Kresse, I., Maldonado, A., Mösenlechner, L., Pangercic, D., Rühr, T., and Tenorth, M. "Robotic Roommates Making Pancakes". In: *Proc. of the IEEE-RAS International Conference on Humanoid Robots*. Bled, Slovenia, Oct. 2011.

Bellman, R. "A Markovian Decision Process". In: *Journal of Mathematics and Mechanics* 6.5 (1957), pp. 679–684.

Bellman, R. *Dynamic Programming*. 1st ed. Princeton, NJ, USA: Princeton University Press, 1957.

Berg, J. van den, Guy, S., Lin, M., and Manocha, D. "Reciprocal n-Body Collision Avoidance". In: *International Journal of Robotics Research*. Vol. 70. 2011, pp. 3–19.

Berg, J. van den, Lin, M., and Manocha, D. "Reciprocal Velocity Obstacles for real-time multi-agent navigation". In: *Proc. of the IEEE International Conference on Robotics and Automation*. 2008.

Bernstein, D. S., Givan, R., Immerman, N., and Zilberstein, S. "The Complexity of Decentralized Control of Markov Decision Processes". In: *Mathematics of Operations Research* 27.4 (2002), pp. 819–840.

Bischoff, R., Huggenberger, U., and Prassler, E. "Kuka youbot-a mobile manipulator for research and education". In: *Proc. of the IEEE International Conference on Robotics and Automation*. IEEE. 2011, pp. 1–4.

Bjarnason, R., Fern, A., and Tadepalli, P. "Lower Bounding Klondike Solitaire with Monte-Carlo Planning." In: *Proc. of the International Conference on Automated Planning and Scheduling*. 2009.

Boutilier, C. "Planning, learning and coordination in multiagent decision processes". In: *Proc. of the 6th Conference on Theoretical Aspects of Rationality and Knowledge*. 1996, pp. 195–210.

Browne, C. B., Powley, E., Whitehouse, D., Lucas, S. M., Cowling, P. I., Rohlfshagen, P., Tavener, S., Perez, D., Samothrakis, S., and Colton, S. "A survey of monte carlo tree search methods". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.1 (2012), pp. 1–43.

Bruce, J. and Veloso, M. "Safe Multirobot Navigation Within Dynamics Constraints". In: *Proceedings of the IEEE* 94.7 (July 2006), pp. 1398–1411.

Brutschy, A., Pini, G., Pinciroli, C., Birattari, M., and Dorigo, M. "Self-organized task allocation to sequentially interdependent tasks in swarm robotics". In: *Autonomous Agents and Multi-Agent Systems* 28.1 (2014), pp. 101–125.

Calliess, J.-P., Lyons, D., and Hanebeck, U. D. "Lazy auctions for multi-robot collision avoidance and motion control under uncertainty". In: *Advanced Agent Technology*. Springer, 2012, pp. 295–312.

Capitán, J., Spaan, M. T. J., Merino, L., and Ollero, A. "Decentralized Multi-robot Cooperation with Auctioned POMDPs". In: *International Journal of Robotics Research* 32.6 (2013), pp. 650–671.

Chazelle, B. "On the convex layers of a planar set". In: *IEEE Transactions on Information Theory* 31.4 (1985), pp. 509–517.

Choi, H.-l., Brunet, L., and How, J. P. "Consensus-based decentralized auctions for robust task allocation". In: *IEEE Transactions on Robotics* (2009).

Claes, D., Hennes, D., Tuyls, K., and Meeussen, W. "Collision avoidance under bounded localization uncertainty". In: *Proc. of the International Conference on Intelligent Robots and Systems*. Oct. 2012, pp. 1192–1198.

Claes, D., Robbel, P., Oliehoek, F. A., Tuyls, K., Hennes, D., and Hoek, W. van der. "Effective approximations for multi-robot coordination in spatially distributed tasks". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2015, pp. 881–890.

Coulom, R. "Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search". In: *Computers and Games: 5th International Conference, CG 2006, Turin, Italy, May 29-31, 2006. Revised Papers*. Ed. by H. J. van den Herik, P. Ciancarini, and H. H. L. M. (Donkers. Berlin, Heidelberg: Springer Berlin Heidelberg, 2007, pp. 72–83.

De Hauwere, Y.-M., Vrancx, P., and Nowé, A. "Learning multi-agent state space representations". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2010, pp. 715–722.

Dias, M. B., Zlot, R., Kalra, N., and Stentz, A. "Market-based multirobot coordination: A survey and analysis". In: *Proceedings of the IEEE* 94.7 (2006), pp. 1257–1270.

Dibangoye, J. S., Amato, C., Buffet, O., and Charpillet, F. "Exploiting Separability in Multiagent Planning with Continuous-State MDPs". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2014, pp. 1281–1288.

Dibangoye, J. S., Amato, C., and Doniec, A. "Scaling Up Decentralized MDPs Through Heuristic Search". In: *Proc. of the Conference on Uncertainty in Artificial Intelligence*. 2012, pp. 217–226.

Doshi, F. and Roy, N. "The permutable POMDP: fast solutions to POMDPs for preference elicitation". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2008, pp. 493–500.

Droge, G. and Egerstedt, M. "Adaptive look-ahead for robotic navigation in unknown environments." In: *Proc. of the International Conference on Intelligent Robots and Systems*. 2011, pp. 1134–1139.

Enright, J. and Wurman, P. R. "Optimization and Coordinated Autonomy in Mobile Fulfillment Systems." In: *Automated action planning for autonomous mobile robots*. 2011, pp. 33–38.

Ferrara, A. and Rubagotti, M. "A dynamic obstacle avoidance strategy for a mobile robot based on sliding mode control". In: *IEEE Control Applications (CCA) and Intelligent Control (ISIC)*. July 2009, pp. 1535–1540.

Fiorini, P. and Shiller, Z. "Motion planning in dynamic environments using velocity obstacles". In: *International Journal of Robotics Research* 17 (7 1998), pp. 760–772.

Flynn, A. and Vencat, E. F. *Custom Nation: Why Customization is the Future of Business and how to Profit from it*. BenBella Books, 2012.

Ford. *Motel T Facts*. Ed. by ford.com. Aug. 2013. URL: https://media.ford.com/content/fordmedia/fna/us/en/news/2013/08/05/model-t-facts.html.

Fox, D., Burgard, W., D., F., and Thrun, S. "Monte Carlo Localization: Efficient Position Estimation for Mobile Robots". In: *Proc. of the AAAI Conference on Artificial Intelligence*. 1999.

Fox, D., Burgard, W., and Thrun, S. "The dynamic window approach to collision avoidance". In: *IEEE Robotics Automation Magazine* 4.1 (Mar. 1997), pp. 23–33.

Fox, D. "Adapting the Sample Size in Particle Filters Through KLD-Sampling". In: *International Journal of Robotics Research* 22 (2003).

Gale, D. *The theory of linear economic models*. University of Chicago press, 1960.

Gelly, S. and Silver, D. "Combining online and offline knowledge in UCT". In: *Proc. of the International Conference on Machine Learning*. 2007, pp. 273–280.

Gerkey, B. P. and Mataric, M. J. "Multi-robot task allocation: Analyzing the complexity and optimality of key architectures". In: *Proc. of the IEEE International Conference on Robotics and Automation*. Vol. 3. IEEE. 2003, pp. 3862–3868.

Gerkey, B. P. and Mataric, M. J. "Sold!: Auction methods for multirobot coordination". In: *IEEE Transactions on Robotics and Automation* 18.5 (2002), pp. 758–768.

Gerkey, B. P. and Matarić, M. J. "A formal analysis and taxonomy of task allocation in multi-robot systems". In: *The International Journal of Robotics Research* 23.9 (2004), pp. 939–954.

Gerkey, B. P., Vaughan, R. T., and Howard, A. "The Player/Stage Project: Tools for Multi-Robot and Distributed Sensor Systems". In: *Proc. of the International Conference on Advanced Robotics*. 2003, pp. 317–323.

Gerkey, B. P. "On multi-robot task allocation". PhD thesis. University of Southern California, 2003.

Gmytrasiewicz, P. J. and Doshi, P. "A Framework for Sequential Planning in Multi-Agent Settings". In: *Journal of Artificial Intelligence Research* 24 (2005), pp. 49–79.

Gordon, G. J., Varakantham, P., Yeoh, W., Lau, H. C., Aravamudhan, A. S., and Cheng, S. "Lagrangian Relaxation for Large-Scale Multi-agent Planning". In: *Proc. of the*

IEEE/WIC/ACM International Conferences on Intelligent Agent Technology, IAT. 2012, pp. 494–501.

Grisetti, G., Stachniss, C., and Burgard, W. "Improved techniques for grid mapping with rao-blackwellized particle filters". In: *IEEE Transactions on Robotics* 23 (2007), pp. 43–46.

Guéant, O., Lasry, J.-M., and Lions, P.-L. "Mean Field Games and Applications". In: *Paris-Princeton Lectures on Mathematical Finance 2010*. Vol. 2003. Lecture Notes in Mathematics. Springer Berlin Heidelberg, 2011, pp. 205–266. ISBN: 978-3-642-14659-6.

Guestrin, C., Koller, D., and Parr, R. "Multiagent Planning with Factored MDPs". In: *Advances in Neural Information Processing Systems 14*. 2002, pp. 1523–1530.

Guy, S. J., Chhugani, J., Kim, C., Satish, N., Lin, M. C., Manocha, D., and Dubey, P. "ClearPath: Highly Parallel Collision Avoidance for Multi-Agent Simulation". In: *Symposium on Computer Animation*. 2009.

Hall, M., Eibe, F., Holmes, G., Pfahringer, B., Reutemann, P., and Witten, I. H. "The WEKA data mining software: an update". In: *SIGKDD Explor. Newsl.* 11.1 (2009), pp. 10–18.

Hanna, J. P., Albert, M., Chen, D., and Stone, P. "Minimum cost matching for autonomous carsharing". In: *IFAC-PapersOnLine* 49.15 (2016), pp. 254–259.

Hannover Messe. *Batch size 1, the easy way*. Ed. by hannovermesse.de. Apr. 2015. URL: http://www.hannovermesse.de/en/news/batch-size-1-the-easy-way.xhtml.

Hart, P. E., Nilsson, N. J., and Raphael, B. "A formal basis for the heuristic determination of minimum cost paths". In: *IEEE transactions on Systems Science and Cybernetics* 4.2 (1968), pp. 100–107.

Henn, S., Koch, S., and Wäscher, G. "Order batching in order picking warehouses: a survey of solution approaches". In: *Warehousing in the Global Supply Chain*. Springer, 2012, pp. 105–137.

Hennes, D., Claes, D., Tuyls, K., and Meeussen, W. "Multi-robot collision avoidance with localization uncertainty". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2012.

Hill Jr, F. "The pleasures of "perp dot" products". In: *Graphics gems IV* (1994), pp. 138–148.

Hoey, J., St-Aubin, R., Hu, A. J., and Boutilier, C. "SPUDD: Stochastic Planning using Decision Diagrams". In: *Proc. of the Conference on Uncertainty in Artificial Intelligence*. 1999, pp. 279–288.

Hon, B. E. *Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base*. US Patent 3,876,255. 1975.

Huang, A. S., Olson, E., and Moore, D. C. "LCM: Lightweight communications and marshalling". In: *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE. 2010, pp. 4057–4062.

International Organization for Standardization. *Robots and robotic devices – Safety requirements for industrial robots – Part 1: Robots*. Standard. Geneva, CH: International Organization for Standardization, Mar. 2011.

International Organization for Standardization. *Robots and robotic devices – Safety requirements for industrial robots – Part 2: Robot systems and integration*. Standard. Geneva, CH: International Organization for Standardization, Mar. 2011.

Jansen, D. and Buttner, H. "Real-time Ethernet: the EtherCAT solution". In: *Computing and Control Engineering* 15.1 (2004), pp. 16–21.

Kaelbling, L. P., Littman, M., and Moore, A. "Reinforcement Learning: A Survey". In: *Journal of Artificial Intelligence Research* 4 (1996), pp. 237–285.

Kagermann, H., Lukas, W.-D., and Wahlster, W. "Industrie 4.0: Mit dem Internet der Dinge auf dem Weg zur 4. industriellen Revolution". In: *VDI nachrichten* 13 (2011), p. 11.

Kartal, B., Nunes, E., Godoy, J., and Gini, M. "Monte carlo tree search with branch and bound for multi-robot task allocation". In: *Proc. of the IJCAI-16 Workshop on Autonomous Mobile Service Robots*. 2016.

Kearns, M., Mansour, Y., and Ng, A. Y. "A Sparse Sampling Algorithm for Near-Optimal Planning in Large Markov Decision Processes". In: *Machine Learning* 49.2-3 (2002), pp. 193–208.

Khandelwal, P. and Stone, P. "Multi-Robot Human Guidance: Human Experiments and Multiple Concurrent Requests". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 1369–1377.

Kitano, H., Asada, M., Kuniyoshi, Y., Noda, I., and Osawa, E. *RoboCup: The Robot World Cup Initiative*. 1995.

Kocsis, L. and Szepesvári, C. "Bandit Based Monte-Carlo Planning". In: *Proc. of the European Conference on Machine Learning*. Vol. 4212. Lecture Notes in Computer Science. Springer Berlin / Heidelberg, 2006, pp. 282–293.

Kocsis, L., Szepesvári, C., and Willemson, J. "Improved monte-carlo search". In: *University Tartu, Estonia, Technical Report 1*. 2006.

Koenig, S., Tovey, C., Lagoudakis, M., Markakis, V., Kempe, D., Keskinocak, P., Kleywegt, A., Meyerson, A., and Jain, S. "The power of sequential single-item auctions for agent coordination". In: *Proc. of the National Conference on Artificial Intelligence*. Vol. 21. 2. Menlo Park, CA; Cambridge, MA; London; AAAI Press; MIT Press; 1999. 2006, p. 1625.

Kohlbrecher, S., Meyer, J., Stryk, O. von, and Klingauf, U. "A Flexible and Scalable SLAM System with Full 3D Motion Estimation". In: *Proc. IEEE International Symposium on Safety, Security and Rescue Robotics (SSRR)*. IEEE. Nov. 2011.

Kok, J. R. "Coordination and learning in cooperative multiagent systems". PhD thesis. University of Amsterdam, May 2006.

Koren, Y. and Borenstein, J. "Potential field methods and their inherent limitations for mobile robot navigation". In: *Proc. of the IEEE International Conference on Robotics and Automation*. Apr. 1991, 1398–1404 vol.2.

Kraetzschmar, G. K., Hochgeschwender, N., Nowak, W., Hegger, F., Schneider, S., Dwiputra, R., Berghofer, J., and Bischof, R. "RoboCup@Work: Competing for the Factory of the Future". In: *RoboCup Symposium*. 2014.

Kruse, T., Pandey, A. K., Alami, R., and Kirsch, A. "Human-aware robot navigation: A survey". In: *Robotics and Autonomous Systems* 61.12 (2013), pp. 1726–1743.

Kuhn, H. W. "The Hungarian method for the assignment problem". In: *Naval Research Logistics (NRL)* 2.1-2 (1955), pp. 83–97.

Lemmens, N. and Tuyls, K. "Stigmergic Landmark Optimization". In: *Advances in Complex Systems* 15.8 (2012).

Linder, T., Breuers, S., Leibe, B., and Arras, K. O. "On multi-modal people tracking from mobile platforms in very crowded and dynamic environments". In: *Proc. of the IEEE International Conference on Robotics and Automation*. IEEE. 2016, pp. 5512–5519.

Lu, D. V. "Contextualized Robot Navigation". PhD thesis. Washington University, Dec. 2014.

MacAlpine, P., Price, E., and Stone, P. "SCRAM: Scalable collision-avoiding role assignment with minimal-makespan for formational positioning". In: *Proceedings of the 2014 international conference on Autonomous agents and multi-agent systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 1463–1464.

Matignon, L., Jeanpierre, L., and Mouaddib, A.-I. "Coordinated Multi-Robot Exploration Under Communication Constraints Using Decentralized Markov Decision Processes". In: *Proc. of the AAAI Conference on Artificial Intelligence*. 2012, pp. 2017–2023.

McCarthy, J. *An introduction to theoretical kinematics*. Cambridge, Mass. MIT Press, 1990.

Melo, F. S. and Veloso, M. "Learning of coordination: exploiting sparse interactions in multiagent systems". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2009, pp. 773–780.

Nanjanath, M. and Gini, M. "Repeated auctions for robust task execution by a robot team". In: *Robotics and Autonomous Systems* 58.7 (2010), pp. 900–909.

Niemueller, T., Ewert, D., Reuter, S., Ferrein, A., Jeschke, S., and Lakemeyer, G. "RoboCup logistics league sponsored by Festo: a competitive factory automation testbed". In: *Automation, Communication and Cybernetics in Science and Engineering 2015/2016*. Springer, 2016, pp. 605–618.

Niemueller, T., Reuter, S., Ewert, D., Ferrein, A., Jeschke, S., and Lakemeyer, G. "Decisive factors for the success of the carologistics robocup team in the robocup logistics league 2014". In: *Robot Soccer World Cup*. Springer. 2014, pp. 155–167.

Nijssen, P. and Winands, M. H. "Monte carlo tree search for the hide-and-seek game Scotland Yard". In: *IEEE Transactions on Computational Intelligence and AI in Games* 4.4 (2012), pp. 282–294.

Nunes, E., Manner, M., Mitiche, H., and Gini, M. "A taxonomy for task allocation problems with temporal and ordering constraints". In: *Robotics and Autonomous Systems* 90 (2017), pp. 55–70.

Osten, F. B. von der, Kirley, M., and Miller, T. "Anticipatory stigmergic collision avoidance under noise". In: *Proc. of the conference on Genetic and evolutionary computation*. ACM. 2014, pp. 65–72.

Pan, Z., Polden, J., Larkin, N., Van Duin, S., and Norrish, J. "Recent progress on programming methods for industrial robots". In: *Robotics and Computer-Integrated Manufacturing* 28.2 (2012), pp. 87–94.

Pratt, G. and Manzo, J. "The DARPA robotics challenge [competitions]". In: *IEEE Robotics & Automation Magazine* 20.2 (2013), pp. 10–12.

Puterman, M. L. *Markov Decision Processes—Discrete Stochastic Dynamic Programming*. John Wiley & Sons, Inc., 1994.

Quigley, M., Conley, K., Gerkey, B. P., Faust, J., Foote, T., Leibs, J., Wheeler, R., and Ng, A. Y. "ROS: An open-source Robot Operating System". In: *Proc. of the Open-Source Software Workshop (ICRA)*. 2009.

Rios-Martinez, J., Renzaglia, A., Spalanzani, A., Martinelli, A., and Laugier, C. "Navigating between people: A stochastic optimization approach". In: *Proc. of the IEEE International Conference on Robotics and Automation*. IEEE. 2012, pp. 2880–2885.

Saska, M., Mejia, J. S., Stipanović, D. M., Vonásek, V., Schilling, K., and PNV reuNV cil, L. "Control and navigation in manoeuvres of formations of unmanned mobile vehicles". In: *European Journal of Control* 19.2 (2013), pp. 157–171.

Scharpff, J., Roijers, D. M., Oliehoek, F. A., Spaan, M. T., and Weerdt, M. M. de. "Solving Transition-Independent Multi-Agent MDPs with Sparse Interactions." In: *AAAI*. 2016, pp. 3174–3180.

Scharpff, J., Spaan, M. T., Volker, L., and De Weerdt, M. "Planning under Uncertainty for Coordinating Infrastructural Maintenance." In: *ICAPS*. 2013.

Schillinger, P., Kohlbrecher, S., and Stryk, O. von. "Human-Robot Collaborative High-Level Control with an Application to Rescue Robotics". In: *Proc. of the IEEE International Conference on Robotics and Automation*. Stockholm, Sweden, May 2016.

Schneider, E., Balas, O., Ozgelen, A. T., Sklar, E. I., and Parsons, S. "Evaluating auction-based task allocation in multi-robot teams". In: *Proc. of the AAMAS Workshop: ARMS*. 2014.

Schneider, E., Sklar, E. I., and Parsons, S. "Evaluating Multi-Robot Teamwork in Parameterised Environments". In: *Proc. of the Conference Towards Autonomous Robotic Systems*. Springer. 2016, pp. 301–313.

Schneider, E., Sklar, E. I., Parsons, S., and Özgelen, A. T. "Auction-based task allocation for multi-robot teams in dynamic environments". In: *Proc. of the Conference Towards Autonomous Robotic Systems*. Springer. 2015, pp. 246–257.

Schneider, J. G., Wong, W.-K., Moore, A. W., and Riedmiller, M. A. "Distributed Value Functions". In: *Proc. of the International Conference on Machine Learning*. 1999, pp. 371–378.

Schneider, S., Hegger, F., Hochgeschwender, N., Dwiputra, R., Moriarty, A., Berghofer, J., and Kraetzschmar, G. K. "Design and development of a benchmarking testbed for the Factory of the Future". In: *Proc. of the IEEE Conference on Emerging Technologies & Factory Automation (ETFA)*. IEEE. 2015, pp. 1–7.

Seuken, S. and Zilberstein, S. "Formal models and algorithms for decentralized decision making under uncertainty". In: *Autonomous Agents and Multi-Agent Systems* 17.2 (2008), pp. 190–250.

Seward, C. *Accelerating Warehouse Operations with Neural Networks*. https://tech.zalando.de/blog/accelerating-warehouse-operations-with-neural-networks/. 2015.

Shapley, L. S. and Shubik, M. "The assignment game I: The core". In: *International Journal of game theory* 1.1 (1971), pp. 111–130.

Shoham, Y. and Tennenholtz, M. "On social laws for artificial agent societies: off-line design". In: *Artificial Intelligence* 73.1–2 (1995), pp. 231–252.

Silver, D. and Veness, J. "Monte-Carlo Planning in Large POMDPs". In: *Advances in Neural Information Processing Systems 23*. 2010, pp. 2164–2172.

Silver, D. et al. "Mastering the game of Go with deep neural networks and tree search". In: *Nature* 529 (2016), pp. 484–503.

Sleight, J. and Durfee, E. H. "A decision-theoretic characterization of organizational influences". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2012, pp. 323–330.

Snape, J., Berg, J. P. van den, Guy, S. J., and Manocha, D. "The Hybrid Reciprocal Velocity Obstacle". In: *IEEE Transactions on Robotics* (2011).

Snape, J., Berg, J. van den, Guy, S. J., and Manocha, D. "Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles". In: *Proc. of the International Conference on Intelligent Robots and Systems*. 2009.

Spaan, M. T. J. and Melo, F. S. "Interaction-Driven Markov Games for Decentralized Multiagent Planning under Uncertainty". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. 2008, pp. 525–532.

Sukthankar, G., Geib, C., Bui, H. H., Pynadath, D., and Goldman, R. P. *Plan, activity, and intent recognition: Theory and practice*. Newnes, 2014.

Sutton, R. S. and Barto, A. G. *Reinforcement Learning: An Introduction*. The MIT Press, Mar. 1998.

Sutton, R. S. "Temporal credit assignment in reinforcement learning". PhD thesis. University of Massachusetts Amherst, 1984.

Theraulaz, G. and Bonabeau, E. "A brief history of stigmergy". In: *Artificial life* 5.2 (1999), pp. 97–116.

Thrun, S., Burgard, W., and Fox, D. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

Tuyls, K., Alers, S., Cucco, E., Claes, D., and Bloembergen, D. "A Telepresence-Robot Approach for Efficient Coordination of Swarms". In: *Proc. of the International Conference on the Synthesis and Simulation of Living Systems (Alife)*. MIT Press, 2016.

Varakantham, P., Cheng, S., Gordon, G. J., and Ahmed, A. "Decision Support for Agent Populations in Uncertain and Congested Environments". In: *Proc. of the AAAI Conference on Artificial Intelligence*. 2012.

Vaughan, R. "Massively multi-robot simulation in stage". In: *Swarm Intelligence* 2.2 (2008), pp. 189–208.

Watkins, C. J. C. H. "Learning from delayed rewards". PhD thesis. University of Cambridge England, 1989.

Wired. *A closer look at BMW's production plant*. Ed. by wired.co.uk. Oct. 2014. URL: http://www.wired.co.uk/article/power-plant.

Wu, J. and Durfee, E. H. "Resource-Driven Mission-Phasing Techniques for Constrained Agents in Stochastic Environments". In: *Journal of Artificial Intelligence Research* 38 (2010), pp. 415–473.

Wurman, P. R., D'Andrea, R., and Mountz, M. "Coordinating hundreds of cooperative, autonomous vehicles in warehouses". In: *AI magazine* 29.1 (2008), p. 9.

Zhang, Y. and Parker, L. E. "Task allocation with executable coalitions in multirobot tasks". In: *Proc. of the IEEE International Conference on Robotics and Automation*. 2012, pp. 3307–3314.

List of my Publications

Claes, D. and Tuyls, K. "Multi robot collision avoidance in a shared workspace". In: *Autonomous Robots special Issue on Distributed Robots* (2018), under submission.

Claes, D., Oliehoek, F., Baier, H., and Tuyls, K. "Decentralised Online Planning for Multi-Robot Warehouse Commissioning". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2017, pp. 492–500.

Fossel, J., Schnieders, B., **Claes, D.**, Hennes, D., and Tuyls, K. "NOctoSLAM: Fast Octree Surface Normal Mapping and Registration ". In: *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE. 2017.

Tuyls, K., Alers, S., Cucco, E., **Claes, D.**, and Bloembergen, D. "A Telepresence-Robot Approach for Efficient Coordination of Swarms". In: *The 15th International Conference on the Synthesis and Simulation of Living Systems (Alife)*. MIT Press, 2016.

Broecker, B., **Claes, D.**, Fossel, J., and Tuyls, K. "Winning the RoboCup@ Work 2014 Competition: The smARTLab Approach". In: *RoboCup 2014: Robot World Cup XVIII*. Springer, 2015, pp. 142–154.

Claes, D., Hennes, D., and Tuyls, K. "Towards Human-Safe Navigation with Pro-Active Collision Avoidance in a Shared Workspace". In: *Proceedings of the IROS Workshop on On-line decision-making in multi-robot coordination DEMUR)*. 2015.

Claes, D., Robbel, P., Oliehoek, F. A., Tuyls, K., Hennes, D., and Hoek, W. van der. "Effective Approximations for Multi-Robot Coordination in Spatially Distributed Tasks". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2015, pp. 881–890.

Alers, S., **Claes, D.**, Fossel, J., Hennes, D., and Tuyls, K. "Applied robotics: precision placement in RoboCup@ Work". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 1681–1682.

Alers, S., **Claes, D.**, Tuyls, K., and Weiss, G. "Biologically inspired multi-robot foraging". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2014, pp. 1683–1684.

Alers, S., Tuyls, K., Ranjbar-Sahraei, B., **Claes, D.**, and Weiss, G. "Insect-inspired robot coordination: foraging and coverage". In: *Artificial life* 14 (2014), pp. 761–768.

Claes, D. and Tuyls, K. "Human Robot-Team Interaction". In: *Artificial Life and Intelligent Agents Symposium*. Springer International Publishing. 2014, pp. 61–72.

Alers, S., Bloembergen, D., **Claes, D.**, Fossel, J.-D., Hennes, D., and Tuyls, K. "Telepresence Robots as a Research Platform for AI." In: *AAAI Spring Symposium: Designing Intelligent Robots*. 2013.

Claes, D., Fossel, J., Broecker, B., Hennes, D., and Tuyls, K. "Development of an Autonomous RC-car". In: *Proc. of the International Conference on Intelligent Robotics and Applications*. Springer Berlin Heidelberg. 2013, pp. 108–120.

Claes, D., Robbel, P., Oliehoek, F., Hennes, D., and Tuyls, K. "Effective Approximations for Planning with Spatially Distributed Tasks". In: *Proc. of the Belgian-Dutch Conference on Machine Learning*. Delft University of Technology (TU Delft); under the auspices of the Benelux Association for Artificial Intelligence (BNVKI), the Dutch Research School for Information, and Knowledge Systems (SIKS). 2013.

Fossel, J., Hennes, D., **Claes, D.**, Alers, S., and Tuyls, K. "OctoSLAM: A 3D mapping approach to situational awareness of unmanned aerial vehicles". In: *Proc. of the International Conference on Unmanned Aircraft Systems (ICUAS)*. IEEE. 2013, pp. 179–188.

Hennes, D., **Claes, D.**, and Tuyls, K. "Evolutionary Advantage of Reciprocity in Collision Avoidance". In: *Proc. of the AAMAS Workshop on Autonomous Robots and Multirobot Systems (ARMS)*. 2013.

Claes, D., Hennes, D., Meeussen, W., and Tuyls, K. "CALU: Collision avoidance with localization uncertainty". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2012, pp. 1495–1496.

Claes, D., Hennes, D., and Tuyls, K. "COCALU: Convex outline collision avoidance under localization uncertainty [Demonstration]". In: *Proc. of the Belgian-Dutch Conference on Machine Learning*. 2012, p. 345.

Claes, D., Hennes, D., Tuyls, K., and Meeussen, W. "Collision avoidance under bounded localization uncertainty". In: *Proc. of the International Conference on Intelligent Robots and Systems*. IEEE. 2012, pp. 1192–1198.

Hennes, D., **Claes, D.**, Meeussen, W., and Tuyls, K. "Multi-robot collision avoidance with localization uncertainty". In: *Proc. of the International Joint Conference on Autonomous Agents and Multi Agent Systems*. International Foundation for Autonomous Agents and Multiagent Systems. 2012, pp. 147–154.

Awards and Achievements

- Runner up for the best paper award at AAMAS 2017, São Paolo, Brazil.
- World Champion in the 2014 RoboCup@work competition, João Pessoa, Brazil.
- 1st Place at the 2014 German Open RoboCup@work competition, Magdeburg, Germany.
- Runner up of the best demonstration award at AAMAS 2014, Paris, France.
- World Champion in the 2013 RoboCup@work competition, Eindhoven, the Netherlands.
- Shared 1st Place at the 2013 German Open RoboCup@work competition, Magdeburg, Germany.
- Winner of the best demonstration award at BNAIC 2013, Delft, the Netherlands.
- Runner up for the best paper at BNAIC 2013, Delft, the Netherlands.
- Winner of the best demonstration award at AAMAS 2012, Valencia, Spain.