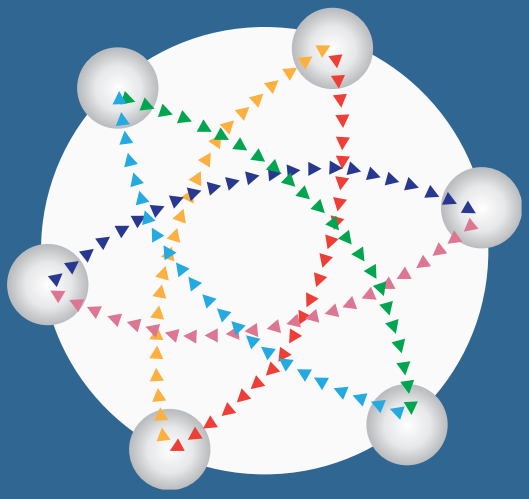


Collision Avoidance

under bounded

Localization Uncertainty

Daniel Claes

Master Thesis

Collision Avoidance under bounded Localization Uncertainty

Daniel Claes

Master Thesis DKE 12-26

Thesis submitted in partial fulfillment
of the requirements for the degree of Master of Science
of Artificial Intelligence at the Department of Knowledge
Engineering of the Maastricht University

Thesis Committee:

Karl Tuyls, Gerhard Weiß and Daniel Hennes
Department of Knowledge Engineering, Maastricht University

Philipp Robbel
Massachusetts Institute of Technology

Maastricht University
Faculty of Humanities and Sciences
Department of Knowledge Engineering
Master Artificial Intelligence

August 17, 2012

Abstract

We present a multi-mobile robot collision avoidance system based on the velocity obstacle paradigm. In contrast to previous approaches, we alleviate the strong requirement for perfect sensing (i.e. global positioning) using Adaptive Monte-Carlo Localization on a per-agent level. While such methods as ClearPath and Optimal Reciprocal Collision Avoidance (ORCA) guarantee local collision-free motion for a large number of robots, given perfect knowledge of positions and speeds, a realistic implementation requires further extensions to deal with inaccurate localization and message passing delays.

The proposed system bounds the error introduced by localization and combines the computation for collision-free motion with localization uncertainty. Current positions and velocities of surrounding robots are translated to an efficient geometric representation to determine safe motions.

In a first approach, the robots and localization uncertainty are approximated by circumscribed radii. This method is further refined using a close convex hull approximation to minimize the overestimation in the footprint and localization uncertainty.

Additionally, methods to deal with static obstacles for the presented approaches are explained, so that a fully distributed local multi-robot navigation solution is provided. Results show that our approach allows for safe navigation even in densely packed and complex environments including static and dynamic obstacles.

Acknowledgments

Even though this section is quite early in the thesis, it was the last one to write. However, it is also one of the most enjoyable section, since it reminds me of the good time I had during my study and the nice people around me. In the last two years many people did a good job in keeping me sane and motivated and I would like to dedicate this section to those that deserve some special attention.

First of all, I would like to thank my supervisors Karl Tuyls and Daniel Hennes. Karl, you already supervised me during my bachelor thesis and both of you supervised me during other projects in my master. Thanks to you, I could avoid the normal project and already work in the swarmlab during my time as master student. I would also like to thank you for your guidance and supervision.

With you Daniel, I went to my first conference, the IROS 2011 in San Francisco and we wrote my first publications together, and we even won the best demonstration award at AAMAS 2012. Besides all of the study, we had a good time at parties, the road trip to spain and other events. So thank you very much for all of this.

I also owe it to my supervisors, Karl and Daniel, that I was given the opportunity to spend an awesome semester abroad at Willow Garage in California. My special thanks to Wim, Troy, Ken and Morten, who made my time at Willow unforgettable. Additionally, I should not forget PR-J, PR-M and the four turtlebots that tried to avoid collisions for me - sometimes successfully, but most of the time not.

I have always felt welcome in the swarmlab and special thanks to all members of the swarmlab for the nice time and atmosphere in the lab. The time is not always spent effectively in the (s)warmlab, but it is always good to be there to take part in the discussions. We had an awesome time at AAMAS 2012 and I hope some more conferences will follow. I am very grateful for Gerhard Weiß and Karl Tuyls to arrange a position for me at the swarmlab so that I can stay here as a PhD candidate.

Many thanks go to Leoni, Bijan and Jörn for proof-reading my thesis and commenting on my sometimes awkward grammar and spelling errors and to Michael to help me with the presentation of the results.

I would also like to express my thanks for my fellow students and house mates. Joscha, we spent already half a year abroad together in Singapore and did various projects together. I could always count on your help to complete

the work, even if that included various night shifts.

Fabian, we lived together for the longest time during my study. It was always good to talk to you and the nights you went to party with us were unforgettable.

Franz, Henning, Andreas and Max, you are also part of the group that "survived" the bachelor and continued in the master with me and our learning sessions in the last week before the exams. Without you guys I would have prepared even less for the exams.

I would like to thank the people of the MaasSAC, which is my one of my biggest distractions during studies. It is always nice to come to the climbing gym. Without you, I would not be able to climb mountains, hike glaciers and reach summits.

Not the last but neither least, I would like to thank my family. Without the help and support of my parents and my brother, I would not have been able to do any of this. It is always nice to come home and feel welcomed as I do.

Volgens de traditie is de laatste bedankte ook de belangrijkste. Renée, zonder jouw steun, hulp en liefde zou ik de laatste tijd niet gehaald hebben. Je moest af en toe een beetje afzien, bij voorbeeld, als ik naar de VS ging, of in de laatste maand van het schrijven van deze scriptie. Toen heb ik niet meer veel anders gedaan als in de studeer kamer te zitten en te schrijven. Maar ook in deze tijd heb je je om mij gezorgd, me eten gebracht en geholpen, onder anderen door de boxen te regelen van je medewerker Karin. Wij konden niet alle "collisies uitwijken", maar gelukkig hebben we geen schade ervan genomen en ook heel vele leuke ervaringen samen, zoals de zomervakantie in de Alpen, Mexico en Noorwegen. Hopelijk komen er nog een aantal meer!

Thank you all!

Contents

List of figures

List of tables

Chapter 1

Introduction

You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist.

Friedrich Nietzsche

1.1 From local collision avoidance to multi-robot collision avoidance

Local collision avoidance is the task of steering free of collisions with static and dynamic obstacles, while following a global plan to navigate towards a goal location. Local collision avoidance differs from motion planning, global path planning and local path planning. In motion planning the environment of the robot is assumed to be deterministic and known in advance, thus allowing to plan a complete path to the goal. Global path planners usually operate on a static map and find either the minimum cost plan (e.g. using A* or Dijkstra's algorithm) or any valid plan (e.g. sample based planners). Local path planners, such as Trajectory Rollout and Dynamic Window Approaches (DWA), perform forward simulations for a set of velocity commands; each resulting trajectory is scored based on proximity to the goal location and a cost map built from current sensor data. In principle this allows to stay clear of dynamical obstacles. However, in multi-robot settings two problems arise:

1. Robots are not just dynamic obstacles; each robot itself is a pro-active agent taking actions to avoid collisions.

2. The sensor source (e.g. laser range finder) is usually mounted on top of the robot's base to allow for a maximal unoccluded viewing angle. In a system with homogeneous robots this implies that there is very little surface area that can be picked up by the sensors of other robots and thus prevents the robots from seeing each other.

Local collision avoidance addresses these challenges and is an important building block in any robot navigation system targeted at multi-robot systems.

Although robot-robot detection 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 without the help of external tools.

1.2 Problem definition

We address the problem of real-time local collision avoidance in multi-mobile robot systems. As explained before, in a multi-mobile robot scenario, robots can not merely be regarded as dynamic obstacles. Each robot is a pro-active agent, taking actions to achieve a certain goal, e.g. to avoid collisions. Neglecting this fact might lead to oscillations and thus highly inefficient trajectories or even collisions.

The velocity obstacle (VO) is a geometric representation of all velocities that will eventually result in a collision given the dynamic obstacle maintains the observed velocity. Van den Berg et al. [30] introduced the reciprocal velocity obstacle (RVO) to address reciprocity: for each pair of robots, one takes half of the responsibility to avoid the other. RVO has since been extended to the hybrid reciprocal velocity obstacle (HRVO) [25, 27] to overcome "reciprocal dances" that occur when robots can not reach independent agreement on which side to pass each other.

The aforementioned approaches avoid local sensing methods by using global positioning via an overhead tracking camera or assume perfect sensing (i.e. purely simulation based), which greatly limits the possibilities for real life applications. Thus, this leads to the following three research goals.

1. We will alleviate the strong requirement for perfect sensing (i.e. global positioning) using adaptive Monte-Carlo localization on a per-agent level. While the VO paradigm guarantees local collision-free motion for a large number of robots, given perfect knowledge of positions and speeds, a realistic implementation requires further extensions to deal with inaccurate localization and message passing delays. We will investigate bounding the error introduced by localization and combining the computation for collision-free motion with localization uncertainty.

2. A robot usually follows a certain goal. In motion planning this can be modeled as each robot having a preferred velocity that the robot wants to pursue in order to get closer to its current goal location. This goal could be waypoints given by a global planning algorithm. In this thesis, we focus on the local collision avoidance task and assume that the preferred velocity is pointing directly towards the goal.
   When using the VO paradigm, there exist multiple ways to select a new

collision free velocity. ORCA, introduced by van den Berg et al. [29], and ClearPath (Guy et al. [17]) are two mathematical optimization methods that search for the collision-free velocity that is closest to the preferred velocity. We will examine how these approach compare to a simple sampling based method.

3. Furthermore, we will investigate on how to incorporate static obstacles in the VO formulation on real robots equipped with a laser range finder (LIDAR) to provide a complete solution that can be used to avoid static as well as dynamic obstacles in a multi-mobile robot scenario.

1.3 Contributions

This section lists concisely the contributions resulting from the research work presented in this thesis.

- Chapter 3 introduces "Collision avoidance with localization uncertainty (CALU)": A novel approach that successfully combines per-agent localization using adaptive Monte-Carlo localization (AMCL) with the velocity obstacle paradigm. The localization uncertainty introduced by sensor and actuator noise is bounded and taken into account for collision avoidance by virtually enlarging the robots' radii. The presented approach is situated in between centralized motion planning for multi-robot systems and communication-free individual navigation. While actions will remain to be computed independently for each robot, information about position, shape and velocity is shared using inter-robot communication. This keeps the communication overhead limited while avoiding problems like robot-robot detection. The approach alleviates the strong requirement for perfect sensing (e.g. global positioning) and results in a robust and distributed system. A paper and demonstration about this chapter are published in the *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS 2012)* [18, 7] respectively.

- In Chapter 4, the previous approach is refined in order to overcome some limitation of CALU. The enlargement of the robots' radii can vastly overestimate the actual size and localization uncertainty of the robots. A closer and error bounded convex approximation of the underlying localization uncertainty distribution is used together with the Minkowski sum of convex hull of the robot's footprint to better approximate the situation. The resulting algorithm "Convex outline collision avoidance under localization uncertainty (COCALU)" is presented and explained. Additionally, a way how to reuse the implementations for circular robots is shown, by reversely fitting a circle in the constructed velocity obstacles such that an equivalent velocity obstacle would have been created. This new circle can be used as relative position and combined radius as in the previous

implementations. A paper about this chapter is published in the *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS 2012)* [8].

- Chapter 5 explains how static obstacles can be integrated in the velocity selection methods. Additionally, it is explained how obstacles can be detected as line-segments using a laser range finder (LIDAR), and how a 3D sensor source (e.g. a Microsoft Kinect) can be transformed to be used with the same line detection algorithm.

- Chapter 6 extensively evaluates and compares the presented approaches in various scenarios. Reasonable parameter settings are investigated that lead to the best results. The methods are tested with up to eight robots in simulation and with up to four robots in real life. In real life, various more complex scenarios including static and dynamic obstacles are further evaluated using the best performing method.

1.4 Related work

The related work in collision avoidance can roughly be divided into two parts; single agent collision avoidance and multi-robot collision avoidance. In previous work, many approaches make use of the "frozen world" assumption, i.e. that the world is static in each timestep. In [28] 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, while around obstacles it is pushing the robot away and near the goal it is pulling the robot towards it. In [21] the limitations of this approach are presented and described. Another approach is the so-called dynamic window approach as described in [14]. However, all of these approaches lack the possibility to navigate safely within a multi-robot environment.

In multi-robot collision avoidance research, there is often a centralized controller. For instance in [4] an approach for safe multi-robot navigation within dynamics constraints is presented. However, these approaches are not robust. If the centralized 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 for instance in [10].

In [3] a method using negotiation is explained. In this case, the robots which approach a collision state have to negotiate the lowest cost paths for both of them. Negotiation relies heavily on communication. This is not desirable, since communication is costly and sometimes error prone, e.g. a lossy wi-fi connection.

In [2] 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 thesis, however the approach

is probabilistic instead of the geometric representation used for the algorithms we propose. Using the geometric approach, we can exploit the shapes and use solution methods such as linear programming for finding the optimal solution for the current state instead of using Monte Carlo sampling.

1.5 Outline

The remainder of this thesis is structured as follows:

Chapter 2 summarizes the approaches used in this thesis. The background on the velocity obstacle paradigm is given and the construction of the various velocity obstacle types (RVO, HRVO) is presented. Additionally, the concept of truncation is explained. Three methods to select a new collision free velocity are shown. Furthermore, the adaptive Monte-Carlo localization method and the Robot Operating System (ROS) are introduced.

Chapter 3 introduces the first algorithm; "Collision avoidance with localization uncertainty (CALU)". The main components of this approach are explained, and in the following, it is shown how the features of the particle filter (AMCL) can be used to derive a bound in the error introduced by localization.

Chapter 4 refines the previous approach to overcome some limitations of CALU. The shortcomings are explained based on the "Corridor Problem", where robots using CALU are not able to move find a solution due to an elongated particle cloud, which largely overestimates the robots' localization uncertainties. The construction of velocity obstacles with convex shapes is revisited and the idea of convex hull peeling with an error bound is presented to get a closer approximation of the actual localization uncertainty. Next, a way to reuse the efficient implementations for circular shaped robots is derived. Finally, it is shown how the new approach "Convex outline collision avoidance under localization uncertainty (COCALU)" solves the corridor problem.

Chapter 5 explains how static obstacles can be integrated in the velocity selection methods. Additionally, it is explained how obstacles can be detected as line-segments using a laser range finder (LIDAR), and how a 3D sensor source (e.g. a Microsoft Kinect) can be transformed to be used with the same line detection algorithm.

Chapter 6 extensively evaluates and compares the presented approaches in various scenarios. The algorithms are tested with up to eight robots in simulation and with up to four robots in real life.

Chapter 7 concludes this thesis.

Chapter 2

Background

This thesis describes a decentralized multi-robot collision avoidance system based on the velocity obstacle paradigm as introduced by Fiorini et al. [11] and per-agent localization. This is in contrast to many other algorithms that utilize centralized planning or assume perfect knowledge about the other robots' positions, shapes and speeds.

The first section of this chapter focuses on the construction of the various types of the velocity obstacles that have evolved over time to take reciprocality 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. In Section 2.5 adaptive Monte-Carlo localization (AMCL) is explained and finally, the Robot Operating System (ROS) is introduced.

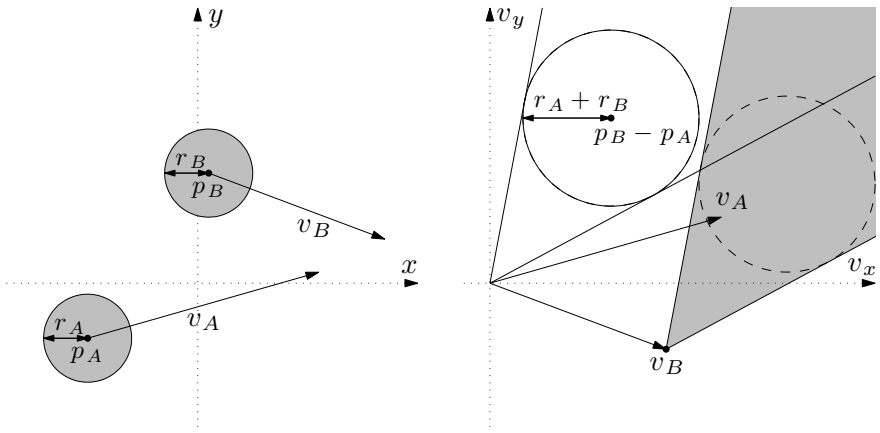
2.1 Introduction

In an environment with only static obstacles, traditional planning algorithms can be used. However, dynamic obstacles pose a tough challenge.

The velocity obstacle (VO) was introduced as one approach to deal with dynamic obstacles. The VO is a geometric representation of all velocities that will eventually result in a collision given 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.

The velocity obstacle paradigm was extended to incorporate reciprocality to the reciprocal velocity obstacle (RVO). This approach assumes that each agents takes half the responsibility for the collision avoidance. This result was further refined to the hybrid reciprocal velocity obstacle (HRVO) to overcome situations in which the reciprocal velocity obstacle could lead to reciprocal dances [20], since the sides on which the robot wants to pass switches with each time step.

If we have calculated the area for each velocity obstacle in the velocity space, we have areas leading to collisions, and collision free velocities. If the preferred



(a) Workspace configuration (b) $VO_{A|B}$

Figure 2.1: (a) A workspace configuration with two robots R_A and R_B on collision course. They are described by a position, radius and velocity (i.e. p_A, r_A and v_A for robot R_A). (b) Translating the workspace configuration into velocity space and the resulting velocity obstacle ($VO_{A|B}$) for R_A. The cone starting at the origin is created by calculating the tangents from the origin to the disc of the combined radii ($r_A + r_B$) at the relative position ($p_B - p_A$). This cone is the velocity obstacle, when the robot R_B would be static. By translating the apex with the other robot's velocity (v_B), the resulting cone is the $VO_{A|B}$.

velocity is leading to a collision, we want to find a velocity that is close to the preferred velocity but still collision free. There are several approaches to calculate this new velocity. Three examples (ORCA, ClearPath and sampling based) will be explained in this chapter.

2.2 Velocity Obstacles (VO)

The velocity obstacle (VO) was introduced for local collision avoidance and navigation in dynamic environments with multiple moving objects. The subsequent definition of the VO assumes planar motions, though the concept extends to 3-D motions in a straight forward manner.

Let us assume a workspace configuration with two robots on a collision course as shown in Figure 2.1(a). If the position and speed of the moving object (robot R_B) is known to R_A, we can mark a region in the robot's velocity space which leads to 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:

$$A \oplus B = \{a + b \mid a \in A, b \in B\} \tag{2.1}$$

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 $atan2$ 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 center p_{rel} as shown in Figure 2.1(b). The angle can then be calculated as:

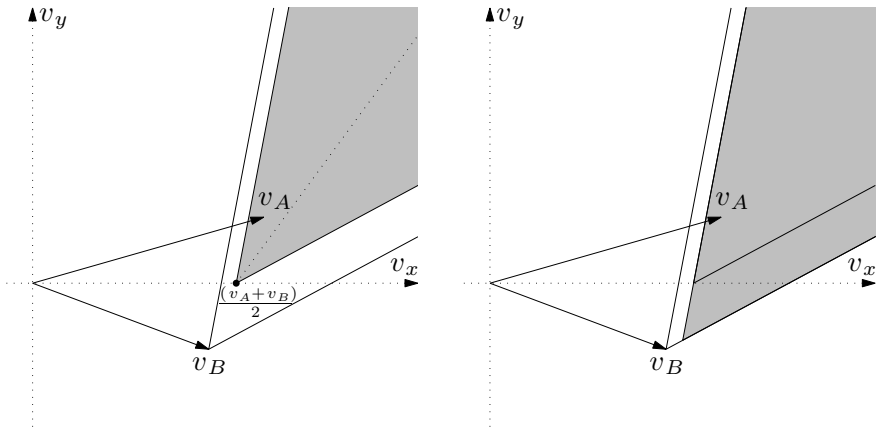
$$\theta_{left} = -\theta_{right} = arcsin(\frac{r_A + r_B}{|p_{rel}|}) \tag{2.4}$$

In the example in Figure 2.1(b), robot R_A's velocity vector v_A points into the VO, thus we know that R_A and R_B are on collision course. Each agent computes a VO for each of the other agents. If all agents at any given time step select velocities outside of the VOs, the trajectories are guaranteed to be collision free. A velocity obstacle induced by R_B for R_A is denoted as $VO_{A|B}$.

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

2.2.1 Reciprocal velocity obstacles (RVO)

To overcome these oscillations, the reciprocal velocity obstacle (RVO) was introduced by Berg et al. [30]. 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 2.2(a). Furthermore, this leads to the property, that if every robot chooses a velocity outside of the RVO closest to the current velocity, the robots will pass on the same side. However, each robot optimizes its commanded velocity in respect to a preferred velocity in order to make progress towards its goal location. This can lead to reciprocal dances, i.e. both robots first try to avoid to the same side and then to the other side. In a situation with perfect symmetry and sensing, this behavior continues infinitely.



(a) $RVO_{A|B}$ (b) $HRVO_{A|B}$

Figure 2.2: (a) Translating the $VO_{A|B}$ by $\frac{v_A+v_B}{2}$ results in the reciprocal velocity obstacle ($RVO_{A|B}$), i.e. each robot has to take care of half of the collision avoidance. (b) Translating the apex of the $RVO_{A|B}$ to the intersection of the closest leg of the $RVO_{A|B}$ to the own velocity, and the leg of the $VO_{A|B}$ 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 velocity obstacle ($HRVO_{A|B}$).

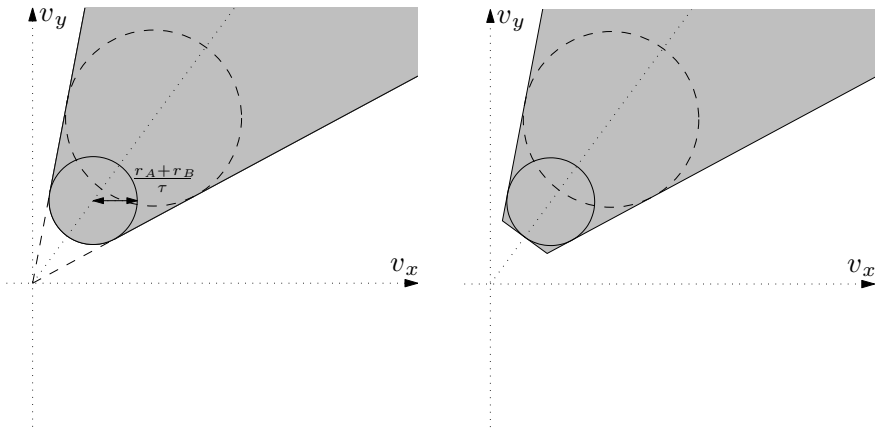
2.2.2 Hybrid reciprocal velocity obstacles (HRVO)

To counter these situations, the hybrid reciprocal velocity obstacle (HRVO) was introduced by Snape et al. [25]. Figure 2.2(b) shows the construction of the HRVO. To encourage the selection of a velocity towards the preferred side, e.g. left in this example, the other 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" side, e.g. another robot induces a HRVO that blocks the whole side. However, all other robots will soon adapt to the new side.

2.2.3 Truncation

When the workspace is cluttered with many robots that do not move or only move slowly, the apexes of the VOs are close to the origin in velocity space; thus rendering the robots immobile. This problem can be solved using truncation.

The idea of a truncating a VO can be best explained by imagining a static obstacle. A velocity in the direction of the obstacle will eventually lead to collision, but not directly. Hence, we can define an area in which the selected



(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.

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 2.3(a). The truncation can be closely approximated by a line perpendicular to the relative position and tangential to the shrunken disk as shown in Figure 2.3(b). VO^τ denotes a truncated velocity obstacle.

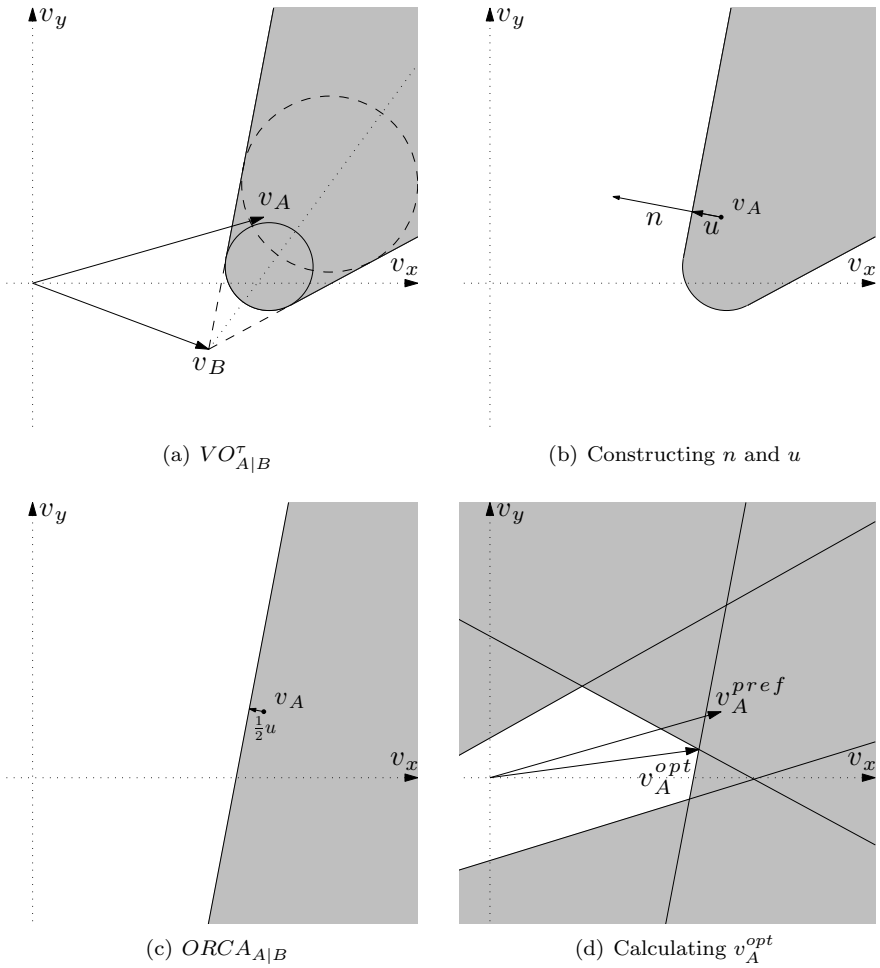
Applying the same method to create a HRVO and RVO from a VO, we can create a truncated HRVO and truncated RVO ($HRVO^\tau$ and RVO^τ) respectively from of VO^τ by translating the apex accordingly.

2.3 Selection of 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 timesteps. Within this region the new velocity has to be selected. In order to do this efficiently, there are several ways to calculate the new velocity. In the following, three ways will be presented.

2.3.1 Optimal Reciprocal Collision Avoidance (ORCA)

To enable efficient calculation for safe velocities, *Optimal Reciprocal Collision Avoidance (ORCA)* was introduced by van den Berg et al. [29]. Instead of velocity obstacles, agents independently compute half-planes of collision-free



(a) $VO^{\tau}_{A|B}$ (b) Constructing n and u

(c) $ORCA_{A|B}$ (d) Calculating v_A^{opt}

Figure 2.4: ORCA. (a) A truncated velocity obstacle ($VO^{\tau}_{A|B}$) for robot R_A. (b) Constructing u and n. Where u is the vector from v_A to the closest boundary of $VO^{\tau}_{A|B}$ and n is the outward normal in the direction of u. (c) Calculating the ORCA half-plane ($ORCA_{A|B}$) perpendicular to u at $v_A + cu$. (d) The velocity space with four ORCA half-planes. v_A^{opt} is the velocity closest to v_A^{pref} that is collision free.

velocities for each other agent. The intersection of all half planes is the set of collision free velocities. The optimal velocity from this set can be calculated by solving a linear program minimizing the distance to the desired goal velocity.

More specifically, let us assume a 2-dimensional workspace with N disk-shaped robots. Each robot $R_i \in N$ has a current position and velocity, $p_i, v_i \in \mathbb{R}^2$. The robots want to reach a certain goal location, and define a desired velocity pointing towards the goal, defined as $v_i^{pref} \in \mathbb{R}^2$. Each robots' objective

is to chose the optimal next velocity command v_i^{opt} such that it is collision free for at least τ time steps and as close as possible to the preferred velocity v_i^{pref}.

The set of collision free velocities can be calculated by constructing the truncated VO (Figure 2.4(a)) for each robot and then looking for the point that is closest to the edge of the VO. The vector from the current velocity towards that point is called u (Figure 2.4(b)). Afterwards a half-plane is constructed perpendicular to the line $v_i^{pref} + c * u$ as shown in Figure 2.4(c). The variable c depicts the amount of responsibility each robot takes for the collision avoidance, where $c = \frac{1}{2}$ means that each robot takes half of the responsibility. $ORCA_{i|j}$ denotes the ORCA half-plane induced by R_j for R_i.

If we have multiple robots, the set of collision free velocities is the intersection of all half-planes. The closest point to the current preferred velocity within the set of collision free velocities is the optimal next velocity command v_i^{opt}. A graphical representation is shown in Figure 2.4(d).

More formally, for the calculation of $ORCA_{i|j}$ the vector to the closest point of the edge of $VOi|j$ is calculated by:

$$u = (\min_{v \in VO_{i|j}} ||v - (v_i - v_j)||) - (v_i - v_j) \tag{2.5}$$

and the vector n is the normal pointing outwards of $VO_{i|j}$ at the point $(v_i - v_j) + u$. Then:

$$ORCA_{i|j} = \{v \in \mathbb{R}^2 | (v - (v_i + c * u)) * n \geq 0\} \tag{2.6}$$

The set of all collision free velocities for R_i is then defined as:

$$ORCA_i = S_{AHV_i} \cap \bigcap_{i \neq j} ORCA_{i|j} \tag{2.7}$$

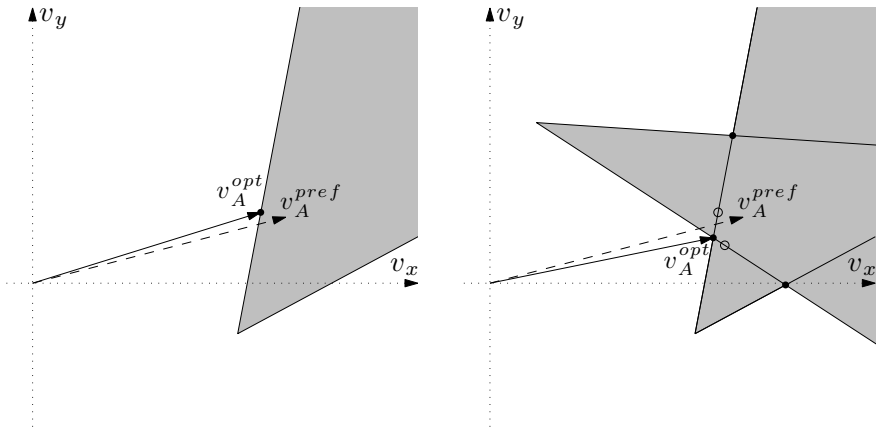
where S_{AHV_i} is the set of allowed holonomic velocities, based on kinematic and motion constraints. The new optimal holonomic velocity command is defined as:

$$v_i^{opt} = \min_{v \in ORCA_i} ||v - v_i^{pref}|| \tag{2.8}$$

This can be solved efficiently in a low dimensional linear program as described in [29].

2.3.2 ClearPath

Another method to efficiently compute collision free velocities, is the ClearPath algorithm introduced by Guy et al. [17]. The algorithm is applicable to many variations of velocity obstacles (VO, RVO or HRVO) represented by line segments or rays, thus for the truncated VOs the approximation with a line is used. ClearPath follows the general idea that the collision free velocity that is closest to 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 2.5 shows the graphical interpretation of the algorithm.



(a) Projection of v_A^{pref} on $VO_{A|B}$ (b) ClearPath

Figure 2.5: (a) The preferred velocity v_A^{pref} is projected on the closest leg of the $VO_{A|B}$. (b) ClearPath enumerates intersection points for all pairs of VOs (solid dots) and projection points (open dots). The point closest to the preferred velocity (dashed line) and outside of all VOs is selected as new velocity (solid line).

2.3.3 Sampling based

Both of the previous described methods are optimal and exhaustive. With ORCA linear programming is used to find the optimal solution for the current situation. ClearPath enumerates all possibilities and selects the point that is closest to the preferred velocity. Another method is to generate achievable sample velocities based on the motion constraints and to test whether these velocities are collision free and how well they are suited. Each samples gets a score according to some properties, e.g. in this case:

1. The distance to the preferred velocity.
2. The distance to the current velocity.
3. If the velocity is within a velocity obstacle or not.
4. The distance to the closest velocity obstacle.

Each value is weighted and added up to a total score. The velocity sample that scored highest will be selected as the next commanded velocity.

2.4 Kinematic and dynamic constraints

Robots can only accelerate and decelerate within certain dynamic constraints. If the acceleration limits and motion model of the robot is known, a region of admissible velocities can be calculated and approximated by a convex polygon.

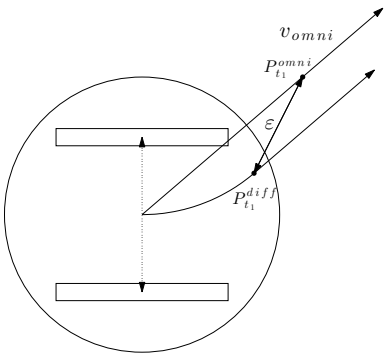


Figure 2.6: The tracking error (ε) is defined as the difference between the position that a holonomic robot would be in after driving with v_omni for t_1 ($P_{t_1}^{omni}$) and the position of the differential drive robot at t_1 ($P_{t_1}^{diff}$).

In order to execute the computed collision free velocity, the robot has to be 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 platform, able to accelerate into any direction from any state. However, differential drive robots with only two motorized wheels are much more common due to their lower price point.

To incorporate the differential drive constraints, Kluge et al. introduced a method to calculate the effective center of a differential drive robot [19]. The effective center represent a translation of the center of rotation to a point that can virtually move into all directions. It can be incorporated in the VO formulation by virtually enlarging the robots' radii prior to the calculations. These adaptations provide additional maneuverability to handle the differential drive constraints. ORCA-DD [26] extends this idea and enlarges the robot to twice the radius of the original size to ensure collision free and smooth paths for robots under differential constraints. The effective center is then located on the circumference of the robot at the center of the extended radius. However, this quadruples the virtual size of the robot, which can result in problems in narrow corridors or unstructured environments.

Another method to handle non-holonomic robot kinematics has been introduced by Alonso-Mora et al [1]: NH-ORCA is the generalized version of ORCA for any non-holonomic robot. However, 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 with a certain tracking error ε. This error depends on the direction and length of the holonomic velocity, i.e. a differential drive robot can drive an arc and then along a straight line which is parallel to a holonomic vector in that direction as shown

in Figure 2.6. 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 v_{omni} for t_1 ($P_{t_1}^{omni}$) 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 radii have to be increased by the tracking error ε, since the robots do not track the desired holonomic velocity exactly. Additionally, in dense configuration with many robots, turn in-place can be included. The set of allowed holonomic (S_{AHV_i}) velocities can be calculated for any possible angle and error. However, any further constraint in the linear program slows down the computation, thus the feasible set can be approximated by a polygon. The velocity space is then constrained by this polygon.

The approach to handle non-holonomic robots given by NH-ORCA is preferred over ORCA-DD, since the virtual increase of the robots' radii is only by a size of ε instead of doubling the radii. Furthermore, it is a general solution that is applicable to any kind of robot that is able to track a holonomic trajectory within a marginal error. To sum up, the velocity space of each robot can be restricted to a set of allowed holonomic velocities (S_{AHV_I}) such that dynamic and kinematic constraints are taken into account.

2.5 Adaptive Monte-Carlo Localization

The localization 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 method, which dynamically adapts the number of particles [12].

Monte-Carlo localization (also known as a particle filter), is a widely applied localization method in the field of mobile robotics. It can be generalized in an initialization phase and two iteratively repeated subsequent phases, the prediction and the update phase.

In the initialization phase, a particle filter generates a number of samples N, which are uniformly distributed over the whole map of possible positions. In the 2.5D case, every particle s^i has a x- and y-value and a rotation $s^i = (\hat{x}, \hat{y}, \hat{\theta})$. The particles are usually initialized in such a way, that only valid positions are taken into account, i.e. they cannot be outside of the map or within walls.

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. 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. In the following subsections, the two phases are explained in further detail.

2.5.1 Prediction phase

After each movement, the position of each particle is updated according to the belief of the agent. More specifically, if the robot has moved forward 10 cm, each particle is moved 10 cm into the direction of its rotation. If the robot rotates, the particles are rotated accordingly. Thus, if a holonomic robot moves from state $\mathbf{x_k} = (x_k, y_k, \theta_k)$ to $\mathbf{x_{k+1}} = (x_{k+1}, y_{k+1}, \theta_{k+1})$, the particles are translated by:

$$\begin{bmatrix} \hat{x}_{k+1} \\ \hat{y}_{k+1} \\ \hat{\theta}_{k+1} \end{bmatrix} = \begin{bmatrix} \hat{x}_k + \rho cos(\hat{\theta}_k + \Delta\theta) \\ \hat{y}_k + \rho sin(\hat{\theta}_k + \Delta\theta) \\ \hat{\theta}_k + \Delta\theta \end{bmatrix} \tag{2.9}$$

Where $\rho = \sqrt{\Delta x^2 + \Delta y^2}$ and $\Delta\theta = \theta_k - \theta_{k+1}$. However, both ρ and $\Delta\theta$ are corrupted by noise due to errors in actuators and odometry. Hence, the more accurate the robot's motion model is, the better the performance of the prediction phase. For non-holonomic robots, the update equations can be changed accordingly [28].

2.5.2 Update phase

After a sensor update, the expected measurement for each particle is calculated. This means, measured sensor values are compared with the world view that is expected if the robot would be at the position of the particle (i.e. by a laser scan matcher). 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:

$$w_{k+1}^i = p(z_k | s_k^i) \tag{2.10}$$

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

$$w_k^i = \frac{w_k^i}{\sum_i w_k^i} \tag{2.11}$$

Particle filters only need a large number of samples N to correctly identify the position when the initial state is unknown. However, when the present localization is quite accurate already, less particles are needed to keep track of the position changes. Hence, the number of samples can be changed adaptively depending on the position uncertainty. We use the approach of KLD-sampling (using the Kullback-Leibler distance), which determines the minimum number of samples needed, such that with probability $1 - \delta$ the error between the true posterior and the sample-based approximation is less than ε. The number of samples can be calculated as:

$$n = \frac{1}{2\varepsilon} \chi^2_{k-1,1-\delta} \tag{2.12}$$

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

Figure 2.7: Typical particle filter situations. (a) A well localized robot at the end of a corridor resulting in a particle cloud with small variance. (b) In an open ended hallway 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.

This can be approximated using the Wilson-Hilferty transformation as:

$$n = \frac{k-1}{2\varepsilon}\left\{1 - \frac{2}{9(k-1)} + \sqrt{\frac{2}{9(k-1)}}z_{1-\delta}\right\} \tag{2.13}$$

where k is the number of bins of the discrete distribution from which the particles are sampled. For further details can be found in [13].

2.5.3 Kidnapped robot and false localization

A problem occurs if there are several locations which are represented similar according to the sensor values. For example two hallways, which have only one door on the right. In these cases, it happens that the robot localizes itself at the wrong position. Furthermore, the robot can be moved by an external force, like a human. This is also known as the kidnapped robot problem. To incorporate sudden changes or wrong localization, a fraction of particles can be moved to a random location. This increases the robustness of the system.

In our work, AMCL is not used for global localization, but rather initialized with a location guess that is within the vicinity of the true position. This enables us to use AMCL for an accurate position tracking without having multiple possible clusters in ambiguous cases.

However, a common problem occurs if the environment looks very similar along the trajectory of the robot, e.g. a long hallway; 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.

On the left, we have a well localized particle cloud that is almost circular, in the middle, the point cloud is elongated in the direction of the corridor,

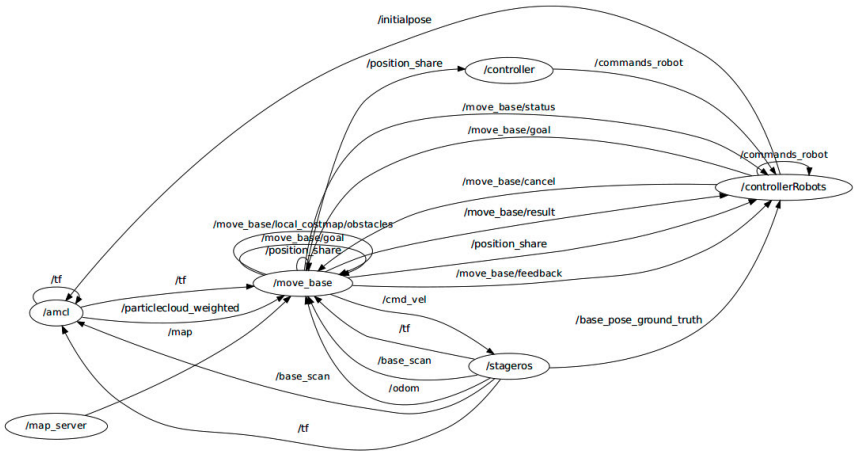


Figure 2.8: Using the ROS visualization tool rxgraph. It is showing the currently running nodes and the connections, i.e. topics shared between the nodes.

since there are only sensor readings on the distance to the walls left and right. The right picture shows the situation in open space, when there are no sensor readings at all. The cloud disperses according to the motion model and is not resampled, since there are no valid sensor readings.

2.6 Robot Operating System (ROS)

The presented algorithms are implemented in the framework of the open source *Robot Operating System (ROS)* [24]. 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"[1].

ROS provides many useful tools, hardware abstraction and a message passing system between nodes. Nodes are self contained modules that run independently and communicate which each other over so called *topics* using a one-to-many subscriber model and the TCP/IP protocol. Naturally this is of great importance when working with distributed systems. However, even in single robot operation, the required functionality can increase immensely, i.e. components for sensor and actuator controls, processing of sensor data together with localization, mapping, navigation and collision avoidance, just to name a few.

Each system has a so-called ROS-master, which is a name server that provides the nodes with the information about which topics are published by which nodes. Thus, each node has to register the topics it publishes at the ROS-master. Another node that wants to subscribe to a topic uses the ROS-master to lookup the address of the publishing on this topics and initiates a direct connection with

[1] For more information see: `http://www.ros.org/`.

them. Hence, after the initial lookup phase, the nodes are connected directly like in a peer-to-peer network.

The software hierarchy can be summarized as follows. Like major operating systems, ROS comes in releases that provide the main functionality and are updated roughly twice a year. To further extend the functionality, stacks are used to bundle many packages together, while in each package there can be several nodes that perform a certain task. These stacks and packages are developed by universities, companies and private people all over the world. A package and stack can be easily shared among the whole community by adding it to an index on the main ROS website.

ROS itself is mainly written in C++ and Python. The ROS-nodes use a client library that is provided for C++, Python, Java, Matlab, Lisp and some other languages. However, any programming language can be used, when the message-passing system and the protocol to communicate with the ROS-master are implemented.

In addition, the modularity enables to easily create various configurations for different settings. The existing modules can be exchanged and hooked up together; in our approach, the same code is used in simulation and on the real robots, only the surrounding modules, e.g. drivers for the sensors, have to be changed. To run our system on any other ROS-enabled robots, only the navigation module needs to be adapted according to the robot's motion and sensor model. The system runs primarily under Ubuntu, while other operating systems are (partially) supported.

Figure 2.8 shows the ROS tool rxgraph for a simulated robot using our system. The tool is useful to see a graph of the currently running nodes (vertexes) and connections (edges) within the system. The graphs are created automatically at runtime, so debugging is facilitated.

To sum up, ROS allows us to reuse many preexisting software packages and drivers. Furthermore, it provides a common framework to enable easy configuration for different types of robots and situations.

2.7 Summary

This chapter has described the theoretical background of the two main principles, the VO-paradigm and AMCL, that are used for the proposed algorithms. For the VO-paradigm, the translation of a workspace situation into a velocity obstacle is described and it is explained how the velocity obstacle can further be shifted to handle reciprocality and to overcome reciprocal dances. Additionally, three methods to select a new velocity are introduced. Furthermore, the Robot Operating System (ROS) is introduced in which the algorithms are implemented.

Chapter 3

Collision avoidance with localization uncertainty (CALU)[1]

We propose a system that builds upon the two main components introduced in Chapter 2, i.e. the VO paradigm and AMCL, to provide collision free motion in a real-world system of robots. In this chapter we will revisit the assumptions commonly made by all velocity-based collision avoidance algorithms and motivate our choice for per agent-based localization in combination with position and velocity information sharing using inter-robot communication. Furthermore, we will point out the necessary addition of sensor uncertainty, leading to our first proposed algorithm *Collision Avoidance with Localization Uncertainty (CALU)*.

3.1 Introduction

VO based algorithms do not require any inter-robot negotiation to find a collision free motion trajectories and are hence in principal fully distributed. However, these methods require perfect information about the positions, velocities and shapes of all other robots. In order to preserve the distributed nature of this approach, robots need to be able to accurately identify other robots using on-board sensors; furthermore, positions and velocities have to be deduced from the same data. The list of typical sensors for mobile robots includes stereo cameras, laser range finders and lately 3D image sensors (e.g. Microsoft

[1] This chapter is based on [7, 18]:
Daniel Claes, Daniel Hennes, Karl Tuyls, and Wim Meeussen. CALU: Collision avoidance with localization uncertainty [Demonstration]. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Valencia, Spain, June 2012
Daniel Hennes, Daniel Claes, Karl Tuyls, and Wim Meeussen. Multi-robot collision avoidance with localization uncertainty. In Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS), Valencia, Spain, June 2012.

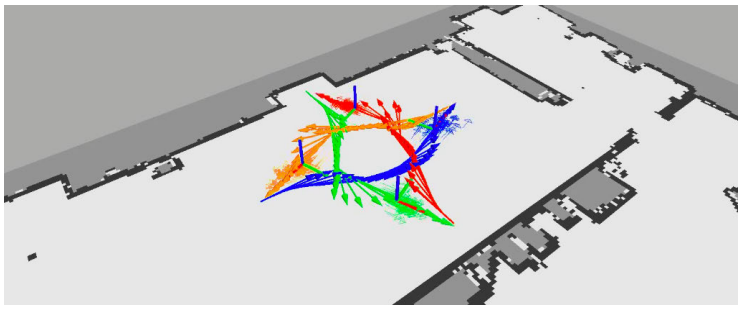


Figure 3.1: CALU with four robots. ROS visualization tool rviz is used to show the trajectories and localization particles of four robots.

Kinect). These sensors deliver large data-streams that require considerable computational power to process even for the detection and classification of static obstacles.

The computational requirement is not the only problem when considering robot-robot detection. As low-end laser range finders (e.g. Hokuyo URG-04LX) become widely available even for mobile robotic projects on a small budget, they are the preferred sensor choice due to their high accuracy, resolution and field of view. However, the laser range finder is usually mounted on top of the robot's base to allow for a maximal unoccluded viewing angle. In a system with homogeneous robots that means that there is very little surface area that can be picked up by the sensors of other robots and thus prevents the robots from observing each other.

Even though the laser range finder provides a high accuracy in the readings, the localization and tracking of the robot using AMCL will in general have the tendency to differ to some extent from the true position of the robot. If the size of the localization and tracking error is in the order of magnitude of the robots radius, collisions are bound to happen.

Previous approaches have worked around these problems by providing global positioning to all robots based on an overhead tracking camera. Such a system is not distributed, since a host computer connected to the camera needs to process the sensor data and communicate with all robots to provide position and velocity data. If this machine fails the system breaks.

3.2 Approach

We propose to utilize agent-based localization and inter-robot communication to provide a system that is more realistic in real-world scenarios (i.e. without the need for external positioning data) and also more robust (i.e. single component failure does not lead to system failure). Our approach, called "collision avoidance with localization uncertainty (CALU)", results in a fully decentralized system that uses local communication to share robot state information in order

to ensure smooth collision free motion; an example for four robots is shown in Figure 3.1. Below we describe the four key components of this approach.

3.2.1 Platform

The robots are assumed to be differential or holonomic drive robots. Required sensors are a laser range finder and wheel odometry. For simplicity we assume a circular footprint; other shapes can be approximated by the circumscribed radius. In order to connect the different subsystems, including device drivers and software modules, we use ROS (see Section 2.6).

3.2.2 Sensor processing and localization

Each robot integrates wheel odometry data which is in turn used to drive the motion model of AMCL (see Section 2.5), hence tracking the pose of the robot. Laser range finder scans are used in the update phase of AMCL. The uncertainty of the current localization, i.e. the spread and weight of the particles, is taken into account for the calculation of collision free velocities as will be explained in further detail in Section 3.3. We assume a prior static map that is used for localization and available to all robots, thus providing a consistent global coordinate frame.

3.2.3 Inter-robot communication

Each robot broadcasts its position and velocity information in the global coordinate frame on a common ROS topic. Each robot also subscribes to the same topic and caches position and velocity data of all other robots. Message delays are taken into account and positions are forward integrated in time according to the motion model of robots using the last known position and velocity information.

3.2.4 Collision avoidance

ORCA, ClearPath or the sampling based approach (see Section 2.3.1) in combination with the constraints for non-holonomic robots (see Section 2.4) can be used to compute collision free velocities according to the aggregated position and velocity data of all surrounding robots. The allowed tracking error for non-holonomic robots is scaled depending on current speed of the robot and the minimal distance to the closest obstacle or other robot. As a last step we incorporate localization uncertainty in the velocity computation as detailed in Section 3.3.

3.3 Localization Uncertainty

The key idea of CALU is to bound the error introduced by localization. To derive this bound, we revisit the particle filter described in Section 2.5.

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:

$$\mu_x = \sum_i w_k^i \; \hat{x}_k^i \tag{3.2}$$

$$\mu_y = \sum_i w_k^i \; \hat{y}_k^i \tag{3.3}$$

$$\mu_\theta = atan2\left(\sum_i w_k^i \; sin(\hat{\theta}_k^i), \sum_i w_k^i \; cos(\hat{\theta}_k^i)\right) \tag{3.4}$$

The mean gives the current position estimate of the robot. However, the estimate is likely to be noisy and we have to take this uncertainty into account in order to ensure collision free motion. The probability of the robot residing within a certain area $\mathcal{A}$ at time k is:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) = \int_{\mathcal{A}} p(\mathbf{x}|\mathbf{z_{1:k}}) d\mathbf{x} \tag{3.5}$$

We can rewrite (3.5) using (3.1) as follows:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \approx \sum_{\forall_i : \mathbf{s}_k^i \in \mathcal{A}} w_k^i \; \delta\left(\mathbf{x}_k - \mathbf{s}_k^i\right) \tag{3.6}$$

From (3.6) we see that for any given $\varepsilon \in [0, 1)$ there is an $\mathcal{A}$ such that:

$$p(\mathbf{x}_k \in \mathcal{A}|\mathbf{z_{1:k}}) \geq 1 - \varepsilon \tag{3.7}$$

Given sufficient samples, the localization uncertainty is thus bounded and we can guarantee that the robot is located within area $\mathcal{A}$ with probability $1 - \varepsilon$.

If a robot radius is inflated by d, the center point of the robot can in turn be translated by a maximum distance of d from its original position while the resulting disc still circumscribes the entire robot. We next derive d such that Equation (3.7) holds.

We define a subset $\mathbf{S} \subset \{\mathbf{s}^1, \ldots, \mathbf{s}^N\}$ with

$$d_{\mathbf{S}} = \max_{(x,y,\theta) \in \mathbf{S}} \left((x - \mu_x)^2 + (y - \mu_y)^2\right) \tag{3.8}$$

the maximal distance to the mean. Furthermore, we define:

$$\mathcal{S} : \mathbf{S} \in \mathcal{S} \text{ iff } p(\mathbf{x}_k \in \mathbf{S}|\mathbf{z_{1:k}}) \geq 1 - \varepsilon$$

There is a minimal subset $\mathbf{S}^* \in \mathcal{S}$ such that (3.7) holds and the maximal distance to the mean is minimized:

$$\mathbf{S}^* = \arg\min_{\mathbf{S}\in\mathcal{S}} d_{\mathbf{S}} \tag{3.9}$$

Thus, if the robot radius is inflated by $d = d_{\mathbf{S}^*}$ the resulting disc circumscribes the entire robot with a probability of $1 - \varepsilon$.

The implementation of this computation is straightforward and efficient. An implementation of AMCL as explained in Section 2.5 commonly tracks particles in a k-d tree structure. The algorithm localizes the node closest to the mean μ and subsequently increases the radius d while adding particles that fall into the radius to the set $\mathbf{S}^*$ and accumulating the weight sum until the threshold $1 - \varepsilon$ is reached.

3.4 Summary

In this chapter, we introduced a decentralized approach for collision avoidance. The localization is per-agent based and local communication is only needed to share position, shape and velocity data. This data could come from sensory data on the robots itself, but this would require additional sensors and image processing, which is beyond the scope of this thesis. Additionally, the assumptions made for this approach are that some form localization is running on any autonomous robot anyway and that in most multi-robot settings local communication should be available.

Furthermore, we introduced an algorithm that bounds error introduced by the localization to a pre-set level. This is necessary to limit the inflation of the radius and thus preventing of rendering the robots immobile. While this approach works well in many cases, there are some limitations that we will address in the following chapter.

Chapter 4

Convex outline collision avoidance under localization uncertainty (COCALU)[1]

In this chapter, we introduce "convex outline collision avoidance under localization uncertainty (COCALU)" as an extension of CALU. In the following section, we will present the corridor problem, which illustrates some of the shortcoming of CALU in more detail. Afterwards, the approach and the algorithm are described, followed by a short discussion about the complexity. The chapter closes with a short summary.

4.1 Introduction

In the previous chapter CALU was introduced. CALU successfully combines per-agent based localization and the VO paradigm to provide collision free motion in a real-world setting with multiple robots. However, some shortcomings prevail. Sub-optimal behavior 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 hallways). In both situations, the resulting VOs vastly overestimate the unsafe velocity regions. Hence, this conservative approximation might lead to a sub-optimal solution - or no solution at all.

COCALU uses the same approach based on decentralized computation, onboard localization and local communication to share relevant shape, position and velocity data between robots. This data is used to build the velocity ob-

[1] This chapter is based on [8]:
Daniel Claes, Daniel Hennes, Karl Tuyls, and Wim Meeussen. Collision avoidance under bounded localization uncertainty. In Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), Algarve, Portugal, October 2012.

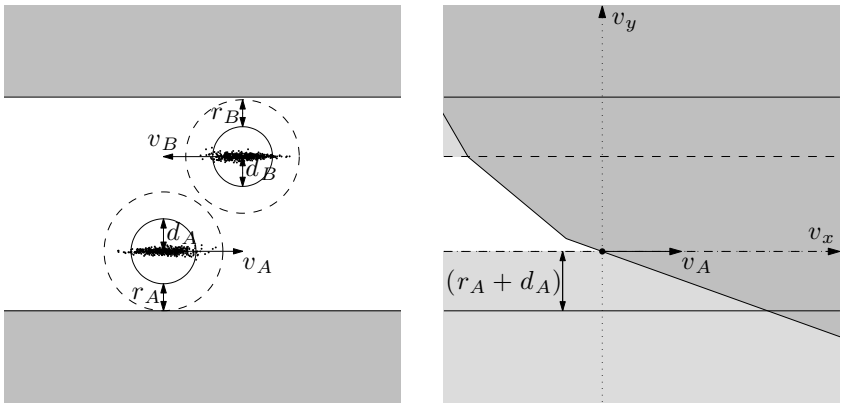


Figure 4.1: The corridor problem: Approximating the localization uncertainty (and the footprint) with circumscribed circles, vastly overestimates the true sizes, such that the robots do not fit next to each other. Thus, the HRVO together with the VO of the walls invalidates all forward movements.

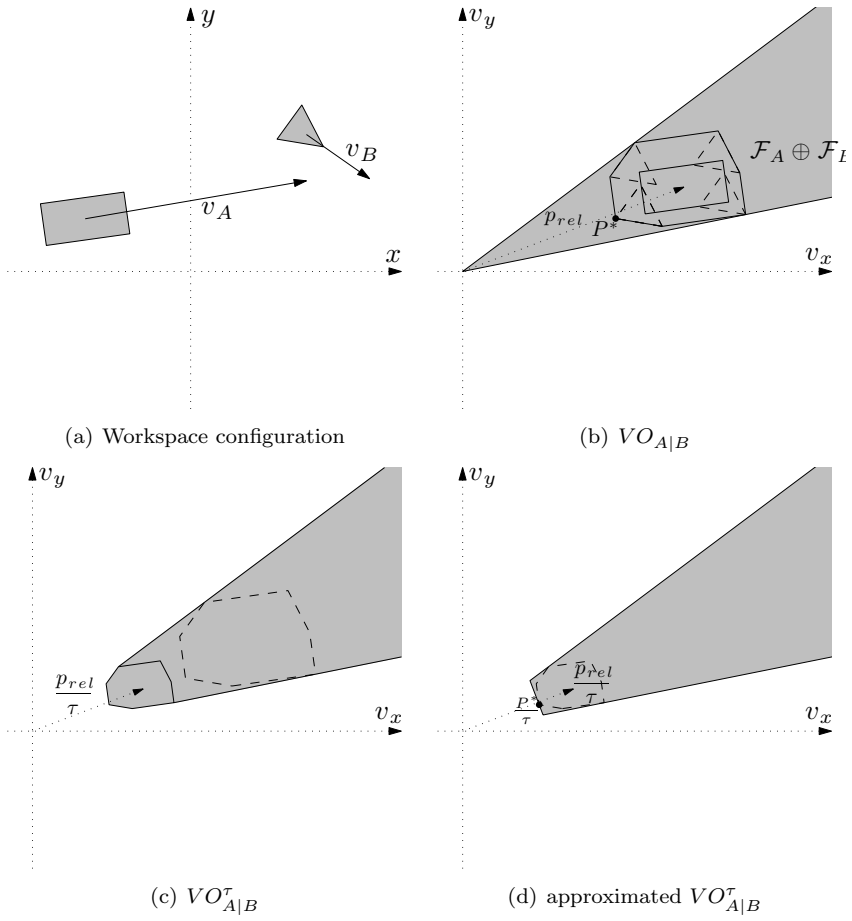
stacle representation using HRVOs for convex footprints in combination with a close and error-bounded convex approximation of the localization density distribution.

4.1.1 Problematic situations with CALU

As mentioned before, a sub-optimal behavior is encountered if the footprint of the robots are vastly overestimated by the circumscribed radius, e.g. in the case of a rectangular robot. Additionally, if the AMCL pose distribution is not shaped circular, but elongated around one axis, or solely based on the motion model of the robot as presented in Figure 2.7, the approximation with a circle is problematic. Figure 4.1 shows an example configuration in which CALU would not be able to move forward, since the two enlarged disks do not fit anymore next to each other in the corridor. In this case, it is due to the elongated position uncertainty, which is largely overestimated by the circle.

4.2 Approach

The key difference between CALU and COCALU is to use the actual shape of the particle cloud instead of using a circumscribed circle. In this approach, we approximate the shape of the particle filter 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, we use "convex hull peeling", which is also known as "onion peeling" [6], in combination with an error bound ε.



(a) Workspace configuration

(b) $VO_{A|B}$

(c) $VO^{\tau}_{A|B}$

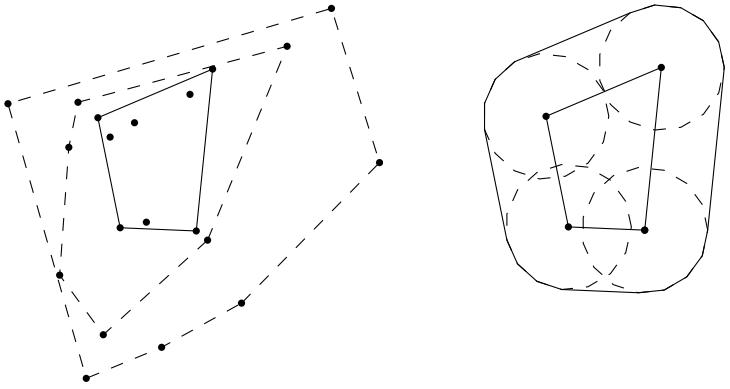
(d) approximated $VO^{\tau}_{A|B}$

Figure 4.2: (a) A workspace configuration with two convex shaped robots. (b) The resulting $VO_{A|B}$, when assuming that the other robot is not moving. (c) Truncating the $VO_{A|B}$. (d) Approximating the $VO^{\tau}_{A|B}$ by a line to limit calculation overhead.

4.3 VOs with convex shapes

In order to understand convex outline VOs as introduced in Section 2.2 in more detail, we will change the example from Figure 2.1(a) to the workspace presented in Figure 4.2(a) with two differently shaped convex outline robots. The $VO_{A|B}$ is obtained by translating the Minkowski sum ($\mathcal{M}$) of the two robot footprints ($\mathcal{F}_A$ and $\mathcal{F}_B$) by the relative position (p_{rel}) and using Equation (2.2) and Equation (2.3) to obtain the angles of the left and right leg respectively as shown in Figure 4.2(b).

To truncate the $VO_{A|B}$, we have to shrink the Minkowski sum $\mathcal{M}$ by the truncation factor τ and translate it by $\frac{p_{rel}}{\tau}$ (Figure 4.2(c)). To facilitate the



(a) Convex hull peeling (b) Minkowski sum

Figure 4.3: Convex hull peeling and Minkowski sum. (a) Three iterations of convex hull peeling. The dotted lines are the convex hulls that removed after each iteration. The solid line is the resulting convex hull after three iterations. (b) Minkowski sum of the resulting convex polygon and a circular footprint.

calculations and decrease the number of borderlines for the $VO^{\tau}_{A|B}$, we can approximate it by the line perpendicular to the relative position and passing through P^* (Figure 4.2(d)). This is defined as the point of the Minkowski sum ($\mathcal{M}$) translated by the relative position (p_{rel}) and closed to the origin:

$$P^* = \min_{P \in \mathcal{M}} |p_{rel} + P| \tag{4.1}$$

As can be seen from Figure 4.2(c) and Figure 4.2(d) the approximation only marginally increases the size of the $VO^{\tau}_{A|B}$. After $VO^{\tau}_{A|B}$ is calculated, the same rules as explained in Section 2.2.1 and Section 2.2.2 apply to calculate the $RVO^{\tau}_{A|B}$ and $HRVO^{\tau}_{A|B}$ respectively.

4.4 Convex hull peeling with an error bound for convex-outline robots

The idea behind the convex hull 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 4.3(a) shows three iterations of the method on an example point cloud.

COCALU finds the convex hull layer in which the probability of the robot being located in is greater than $1 - \varepsilon$. To derive this bound, we revisit the localization uncertainty described in Section 3.3. Given sufficient samples, we can guarantee that the robot is located within area $\mathcal{A}$ with probability $1 - \varepsilon$.

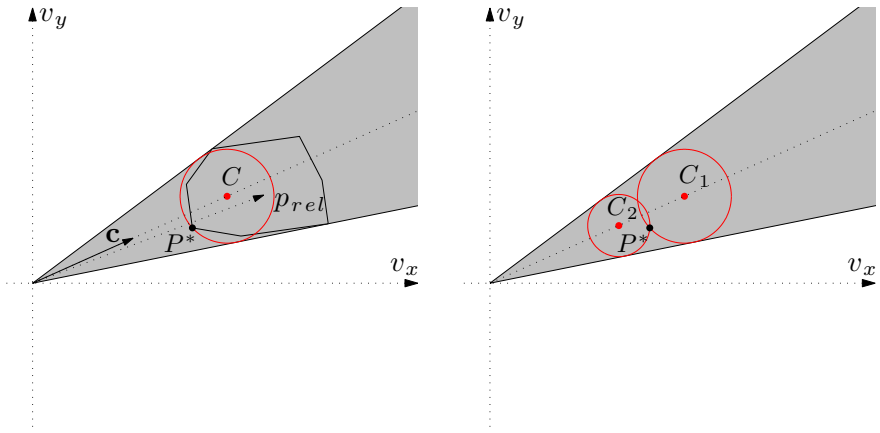
Algorithm 1 COCALU

Input: $(\mathcal{F}_A, p_A, v_A)$: 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 neighboring Agents,
ε : error bound, v_A^{pref} : preferred Velocity,
τ : truncation timesteps

$bound \leftarrow 0$
while $bound \leq \varepsilon$ **do**
  Create convex hull $\mathcal{C}$ of $\mathcal{S}$
  $bound \leftarrow bound + \sum_{\forall i: s^i \in \mathcal{C}} w_i$
  $\mathcal{P} \leftarrow \mathcal{P} \setminus \{(s^i, w^i) \in \mathcal{P} | s^i \in \mathcal{C}\}$
end while
$\mathcal{M}_A \leftarrow \mathcal{F}_A \oplus \mathcal{C}$
for all $(\mathcal{F}_j, p_j, v_j) = A_j \in \mathcal{A}$ **do**
  $\mathcal{M}_j \leftarrow \mathcal{F}_j \oplus \mathcal{M}_A$
  Construct $VO_{A|j}$ from $\mathcal{M}_j$ at $p_j - p_A$
  Construct $VO^{\tau}_{A|j}$ from $VO_{A|j}$ with τ
  if $ORCA$ **then**
    Construct $ORCA_{A|j}$ from $VO^{\tau}_{A|j}$
  else
    Construct $HRVO^{\tau}_{A|j}$ from $VO^{\tau}_{A|j}$ with v_j and v_A
  end if
end for
if $ORCA$ **then**
  Use ORCA linear program to calculate new velocity v_A^{new} from v_A^{pref} and all $ORCA_{A|j}$
else
  ClearPath or the sampling based approach to calculate new velocity v_A^{new} from v_A^{pref} and all $HRVO^{\tau}_{A|j}$
end if

In order to find this specific convex hull enclosing area $\mathcal{A}$, we propose an iterative process as described in the first part in Algorithm 1. 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 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 the 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 4.3(b).



(a) Circular approximation of convex outline (b) Two possible solutions

Figure 4.4: (a) Fitting a circle within the $VO_{A|B}$, such that if the $VO_{A|B}$ would be truncated with $\tau = 1$ still all points of the Minkowski sum would be covered. (b) For a given point P^* that is the closest point to the origin, two circles exist that fulfill the properties. C_1 is the preferred circle, since it leads to the smaller $VO^{\tau}_{A|B}$.

4.5 ORCA with convex shapes

ORCA is well-defined and can be very efficiently implemented for circular robots, but it cannot be used in a straight forward way for convex outline robots. However, using the fact that there always exists a circle that has the two legs of the $VO_{A|B}$ as shown Figure 4.2(b) as tangents such that the resulting $VO^*_{A|B}$ is equivalent to the original $VO_{A|B}$, covering the whole Minkowski sum of the two footprints as shown in Figure 4.4(a). In other words, if we would truncate the $VO_{A|B}$ with $\tau = 1$ with that circle, still the whole Minkowski sum should be covered. When we have calculated this circle, we can reuse the implementation and definitions for circular robots, using the circle's radius as combined radius and the circle's center as relative position.

Starting from the $VO_{A|B}$, we know that the center (C) of the circle is on the line in the middle of the VO and that is has to pass through P^*, which is the closest point of the Minkowski sum to the origin. Hence, we define:

$$\alpha = \frac{\theta_{left} + \theta_{right}}{2},\ \beta = \theta_{rel} + \theta_{left} + \alpha,\ \mathbf{c} = (\cos(\beta), \sin(\beta)),\ C = m * \mathbf{c} \quad (4.2)$$

where θ_{left} and θ_{right} are defined as in Equation (2.2) and Equation (2.3) respectively and θ_{rel} is defined as the angle of the relative position. Then, we know that:

$$\sin(\alpha) = \frac{r}{|C|} \text{ and } |P^* - C| = r \quad (4.3)$$

Squaring both expressions and rewriting the first one leads to:

$$|C|^2 * \sin^2(\alpha) = r^2 \text{ and } |P^* - C|^2 = r^2 \tag{4.4}$$

We can set both left hand sides equal:

$$|C|^2 * \sin^2(\alpha) = |P^* - C|^2 \tag{4.5}$$

We can further rewrite the terms:

$$|C| = |m * \mathbf{c}| = m * |\mathbf{c}| = m \tag{4.6}$$

and:

$$\begin{aligned} |P^* - C|^2 &= (P_x^* - m * \cos(\beta))^2 + (P_y^* - m * \sin(\beta))^2 \\ &= P_x^{*2} - 2 * P_x^* * m * \cos(\beta) + m^2 * \cos^2(\beta) \\ &+ P_y^{*2} - 2 * P_y^* * m * \sin(\beta) + m^2 * \sin^2(\beta) \\ &= P_x^{*2} + P_y^{*2} - 2 * m * (P_x^* * \cos(\beta) + P_y^* * \sin(\beta)) \\ &+ m^2 * (\cos^2(\beta) + \sin^2(\beta)) \\ &= P^* \cdot P^* - 2 * m * \gamma + m^2 \end{aligned} \tag{4.7}$$

with:

$$\gamma = P_x^* * \cos(\beta) + P_y^* * \sin(\beta) \text{ and } (\cos^2(\beta) + \sin^2(\beta)) = 1$$

Replacing the terms leads to:

$$\begin{aligned} & m^2 * \sin^2(\alpha) = P^* \cdot P^* - 2 * m * \gamma + m^2 \\ \Rightarrow & m^2 * \sin^2(\alpha) - m^2 + 2 * m * \gamma = P^* \cdot P^* \\ \Rightarrow & m^2(\sin^2(\alpha) - 1) + 2 * m * \gamma = P^* \cdot P^* \\ \Rightarrow & m^2 + 2 * m * \frac{\gamma}{\sin^2(\alpha) - 1} = \frac{P^* \cdot P^*}{\sin^2(\alpha) - 1} \\ \Rightarrow & \left(m + \frac{\gamma}{\sin^2(\alpha) - 1}\right)^2 = \frac{P^* \cdot P^*}{\sin^2(\alpha) - 1} + \left(\frac{\gamma}{\sin^2(\alpha) - 1}\right)^2 \end{aligned} \tag{4.8}$$

Solving this quadratic equation leads to the following solutions:

$$m_1^* = -\frac{\gamma}{\sin^2(\alpha) - 1} + \sqrt{\frac{P^* \cdot P^*}{\sin^2(\alpha) - 1} + \left(\frac{\gamma}{\sin^2(\alpha) - 1}\right)^2} \tag{4.9}$$

$$m_2^* = -\frac{\gamma}{\sin^2(\alpha) - 1} - \sqrt{\frac{P^* \cdot P^*}{\sin^2(\alpha) - 1} + \left(\frac{\gamma}{\sin^2(\alpha) - 1}\right)^2} \tag{4.10}$$

The two solutions that exist are shown in Figure 4.4(b). Circle C_2 has the properties that we are interested in, but the center is closer to the origin and truncating this VO would lead to a larger over-estimation than using the circle C_1. Thus solution m_1 is the preferred one. There are some special cases, where the Minkowski sum is not completely covered by the calculated VO. This has to be checked and if a point of the Minkowski sum is outside, this point becomes the new P^* and the calculation is iterated again.

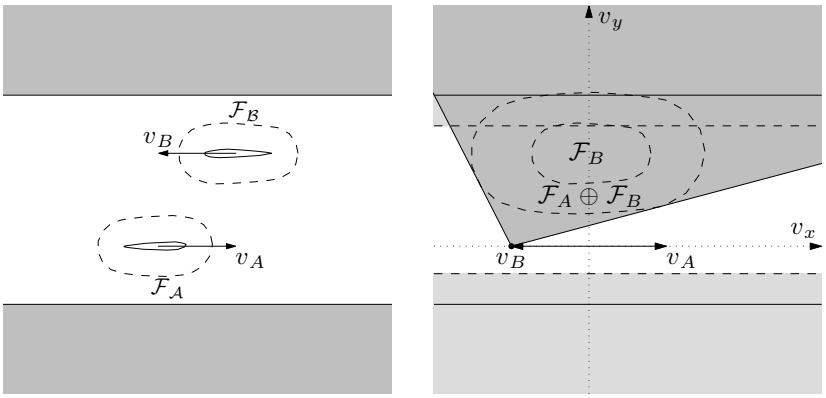


Figure 4.5: Using COCALU solves the corridor problem. Since the robots' footprints and localization uncertainty are approximated with less overestimation, the robots can pass along the corridor without a problem.

4.6 Complexity

The first part of COCALU (see Algorithm 1) computes the convex hull according to the bound ϵ. One iteration of the convex hull can be computed in $\mathcal{O}(n \log h)$ [5], where n is the number of particles and h is the number of points on the hull (our experiments show $h \leq 50$ for $200 \leq n \leq 5000$ particles). The worst case (bound reached in the last iteration) results in complete convex hull peeling, which can be achieved in $\mathcal{O}(n \log n)$ [6].

The convex hull of the Minkowski sum of two convex polygons (operator $\oplus$) can be computed in $\mathcal{O}(l + k)$, where l and k are the number of vertexes (edges). If the input vertexes lists are in order, the edges are sorted by the angle to the x-axis and simply merging the lists results in the convex hull.

ORCA runs in $\mathcal{O}(N)$ [29] and ClearPath runs in $\mathcal{O}(N(N + M))$, where N is the number of VOs and M the number of total intersection segments [17]. The sampling based approach runs in $\mathcal{O}(N * S)$, where S is the number of velocities that are sampled, since we have to check for each sample if it lies within any of the VOs.

4.7 Summary

In summary, using convex hull peeling for approximating localization uncertainty and convex footprints solves the corridor problem. Comparing Figure 4.1 and 4.5 shows the differences when using CALU and COCALU. In the latter figure, it is shown that the robots can easily pass each other without even adapting their path. Furthermore, we introduced a way to reuse the ORCA implementation for circular shapes by approximated the convex $VO_{A|B}$ by the assuming that it would have come from two circular robots. Finally, we have shown that all the calculations are tractable using the complexity analysis.

Chapter 5

Static Obstacles

In the previous chapters we have discussed how the robots can avoid each other in a free-space. However, in a real world setting, more obstacles like walls, chairs and other objects will be present, too. This chapter will introduce how obstacles can be detected from a rotating 2D sensor source, e.g. a laser range finder (also known as LIDAR). Afterwards, it will be explained how data from a 3D camera can be transformed to be used with the same approach as for the LIDAR. Finally, we will show how the detected obstacles can be integrated in the velocity selection methods, i.e. ORCA, ClearPath and the sampling based approach, used in CALU and COCALU.

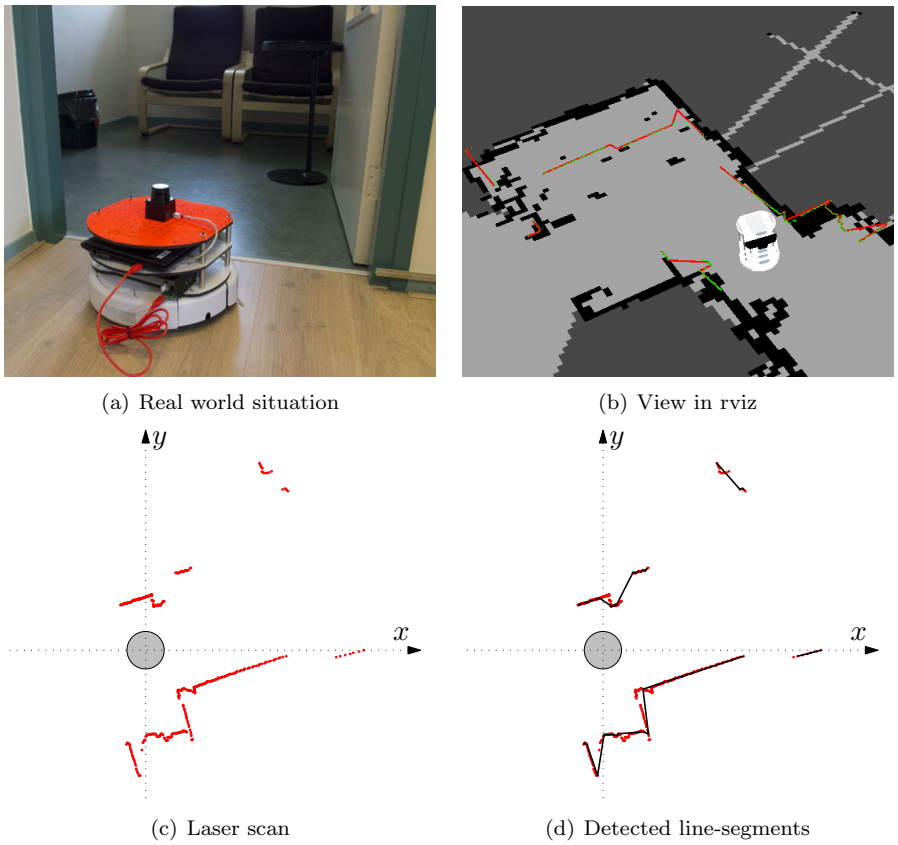
5.1 Introduction

Figure 5.1(a) shows a common situation of an autonomous robot in an office environment. It has a pre-made static map (Figure 5.1(b)), which may or may not include the obstacles seen on the pictures, e.g. the chairs, table, door, etc. The laser range finder is used for localization. Since a laser range finder is a common sensor in robotics, as said before in Section 3.1, we want to use the data also for static obstacle avoidance.

For any obstacle avoidance, it is necessary to detect the outline of the obstacles, in order to avoid them properly. For ORCA-based methods, the obstacles can be modeled as a collection of line-segments [29] and for VO-based methods, the static obstacles can be treated as static robots. In the following, we will explain in more detail how to detect obstacles with a laser range finder and how this can be used with the two proposed algorithms, CALU and COCALU.

5.2 Obstacle detection with a laser range finder

We present a simple and efficient algorithm using a laser scan for obstacle line detection. The algorithm exploits the fact that a laser scan is usually already sorted by increasing angle. If this is not the case, this can be done efficiently. If



(a) Real world situation (b) View in rviz

(c) Laser scan (d) Detected line-segments

Figure 5.1: (a) A real world situation of a robot with a LIDAR and various obstacles. (b) The same situation seen in the ROS visualization tool rviz, with the laser scan and detected line-segments. (c) The laser scan plotted in the frame of the robot. (d) The detected line-segments.

the readings are given in polar coordinates, they can be sorted by the reported angle right away. If the given readings are in Cartesian coordinate we can use the 2D cross product to sort the scan. The 2D cross product is defined as follows:

$$(r_1 - O) \times (r_2 - O) = (x_1 - O_x) * (y_2 - O_y) - (x_2 - O_x) * (y_1 - O_y) \quad (5.1)$$

where $O = (O_x, O_y)$ is the origin and $r_1 = (x_1, y_1)$ and $r_2 = (x_2, y_2)$ are the two readings to compare. If the result is positive, then r_1 is clockwise from r_2, else it is counter-clockwise. When the result is equal to 0, the vectors point in the same direction [9].

Algorithm 2 Line detection from a laser scan

Input: $r_i \in S$: sorted set of laser range readings in Cartesian coordinates
$\varepsilon_{cc}, \varepsilon_{ccs}, \varepsilon_{cvx}, \varepsilon_{cvxs}$: thresholds for concave and convex
r: robot radius
Output: *lines*: a set of line-segments defined with start and end vector

```
i ← 0
while i < length(S) do
  start ← r_i, prev ← r_i, θ_s ← 0, θ_p ← 0
  while true do
    i ← i + 1
    if i > length(S) then
      break
    end if
    next ← r_i
    if Δ(prev − next) > 2 ∗ r then
      break
    end if
    θ_curs ← atan(next − start)
    if prev == start then
      θ_s ← θ_curs
    else
      if (θ_curs − θ_s < ε_ccs) or (θ_curs − θ_s > ε_cvxs) or (θ_curs − θ_p < ε_cc) or
      (θ_curs − θ_p > ε_cvx) then
        i ← i − 1
        break
      end if
    end if
    prev = next, θ_p ← θ_curs
  end while
  lines ← lines ∪ (start, prev)
end while
```

5.2.1 Line detection

Assuming that we have the sorted list, we will go through the points one by one and check if the difference between the angle towards the previous point and the next point exceeds a threshold value or if the difference between the angle towards the first point and the previous point exceed a threshold. As long as that is not the case we continue. Otherwise, we define the line-segment as the first point and the last checked point that fulfilled the requirements and continue with a new line-segment. Figure 5.1(c) shows a laser scan from the situation presented in Figure 5.1(a). In Figure 5.1(d) the detected line-segments are marked. Algorithm 2 shows the implementation of the method.

5.2.2 Dealing with a 3D sensor source

As mentioned before, 3D sensors (e.g. the Microsoft Kinect) become more widely available and are often used in robotics. The advantage of a 3D sensor is that the field of view is not limited to a certain height, such with the laser scan. Hence obstacles which are below the height of the laser scan can be detected. However, a 3D point-cloud coming from such a sensor generates a huge amount of data. This is not feasible to be processed on-board a mobile robot with a low power CPU, e.g. a netbook or a sub-notebook with an ultra-low voltage CPU.

To be able to handle such an amount of data, we can first throttle the scanning speed. For instance a usual 3D sensor runs at 30 Hz, which is not needed, when the controller runs only at a slower speed. Thus the speed of the 3D sensor can be set to the controller frequency, e.g. 10 Hz. Furthermore, the point-cloud can be downsampled by a voxel-grid filter as in [22].

As last step, we can project the throttled and downsampled point-cloud to a 2D plane and generate a fake laser scan out of the data. In order to do this, we define scanning height, a minimal and maximal viewing angle and the resolution of the fake laser. Each possible angle for the laser scan is initialized with a null reading, i.e. the maximum range. For each point in the point-cloud, we project it to the scanning height plane and check the distance of the projected point to the sensor. We calculate the closes angle of the laser scan and chose the corresponding bin. If the current distance is lower than the one currently saved in the bin, we overwrite it, otherwise we neglect the point. This method efficiently transforms the point-cloud into a fake laser scan[1].

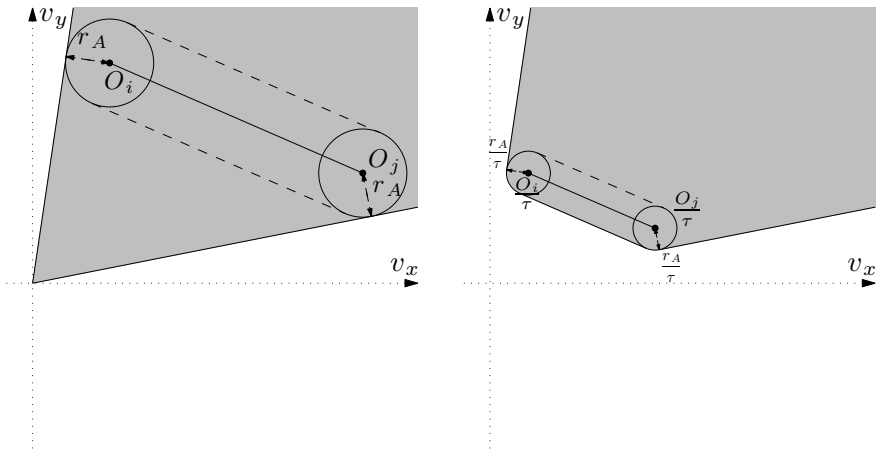
5.3 Static Obstacles with VO-based methods

Static obstacles in VO-based methods can be integrated as if they are static agents. Figure 5.2(a) shows the construction of a VO for a round robot with radius r_A and an obstacle line-segment defined by O_i and O_j. The construction follows the same rules as already presented in Section 2.2.

Since static obstacles, by definition, do not move, we have to truncated 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. Following the same reasoning, we cannot translate the VO, e.g. to create a RVO or HRVO, since these are based on the assumption that the other robot takes part in the collision avoidance, which is not the case for static obstacles.

After the VO^{τ}_{obst} is created, it can be approximated by lines, to make compatible with the ClearPath formulation. The approximation follows the same rules as described in Section 2.2.3 and Section 4.3 for circular and convex outline robots respectively.

[1] For more information and an implementation, see `http://www.ros.org/wiki/pointcloud_to_laserscan`



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

Figure 5.2: (a) Constructing a VO out of a robots' footprint and an obstacle line-segment. (b) Truncating the VO by τ.

5.4 Static Obstacles with ORCA

ORCA uses half-planes and linear programming instead of cones to calculate the new velocity. The computation of the half-planes for other robots is explained in Section 2.3.1. In this section, we will show two possible ways to deal with obstacles, when using the ORCA formulation.

5.4.1 Obstacles as points

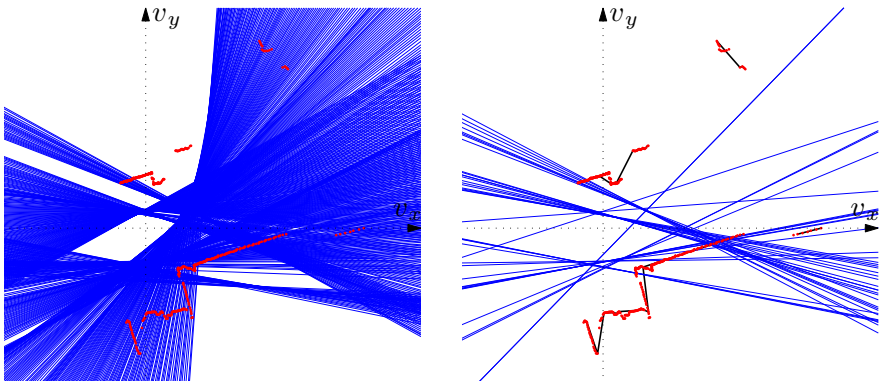
A straight-forward but inefficient way to deal with the obstacles is to use each data point of the laser range finder as an obstacle point. For each point we construct a half-plane perpendicular to the relative position of the robot and the data point. The distance of the half-plane from the origin is defined by:

$$dist = \frac{\min_{p \in \mathcal{F}_i}(p - O)}{\tau} \tag{5.2}$$

where the *min* expression computes the closest distance from any point of the robot's footprint ($\mathcal{F}$) towards the obstacle point (O). This distance is divided by τ to get a truncation similar as explained in Section 2.2.3.

Figure 5.3(a) shows the resulting velocity space, when we construct a half-plane for each data point of the laser range finder. This approach has two main disadvantages:

- For each point an additional line is created. Even though ORCA scales almost linear with number of additional constraints, this is not efficient.



(a) Naïve approach (b) Using line-segments

Figure 5.3: (a) The naïve approach of dealing with obstacles for ORCA. Assuming that every point of the laser scan is an obstacle. This leads to as many additional half-planes as points in the laser scan, e.g. in this case 981. (b) Using the detected line-segments for the half-plane creation. This vastly reduces the amount of half-planes introduced by the obstacles, e.g. in this case down to 36.

- The data points can contain noise. Hence, an outlier within the observations could invalidate the whole velocity space.

5.4.2 Obstacle as line-segments

A second approach is to use the line detection method described in Section 5.2.1. For each detected line-segment, we construct a half-plane. We have two cases for this method.

1. The closest distance from the robot's footprint to the obstacle line-segment is *one of the two obstacle points* defining the obstacle line-segment.

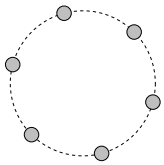
2. The closest distance from the robot's footprint to the obstacle line-segment is *on the line-segment itself.*

In the first case, we can create a half-plane treating the closest point of the obstacle line-segment as a single point and constructing the line as defined in the previous section. In the second case we construct the half-plane parallel to the line-segment, i.e. we construct the half-plane using the point on the obstacle line-segment that is closest to the robot's footprint. For circular robots this point is the projection of the origin on the line-segment. Figure 5.3(b) shows the resulting half planes, depicted as lines. In this example, the number of additional constraints was decreased by more than 95%, when compared to the naïve approach.

5.5 Summary

To conclude, we have shown how a laser range finder can be used for obstacle detection. We exploit the fact that the laser readings are usually ordered with increasing angle, and have also shown how to efficiently order the readings, when this is not the case. A simple but efficient line detection algorithm is explained and how a 3D sensor source can be used to emulate a fake laser.

We have introduced static obstacles for VO-based methods, i.e. ClearPath and sampling based, and ORCA. For the first methods, the implementation of static obstacles is straight forward, since it is basically a not moving agent. For ORCA, we can use each point as an obstacle or use the line detection algorithm. By using the latter one, the number of additional lines can be decreased immensely.



Chapter 6

Evaluation of the methods

This chapter presents experiments and results of the system proposed in the previous chapters. We refine the research goals posed in Section 1.2 and present a common scenario which is used for the evaluation. In the following, we introduce four performance measures and show and discuss the results of the research goals. We begin with parameter tuning in order to determine the best settings for the presented algorithms. Afterwards, the algorithms are tested with various numbers of different robots in simulation and real life. Lastly, we investigate more complex real world scenarios including static and dynamic obstacles. The chapter closes with a short summary.[1]

6.1 Introduction

We have evaluated our approach in simulation using *Stage* [15, 32] and in real-world settings. Simulation allows us to investigate the system performance using many repetition and various extreme settings. For evaluation we have chosen several different scenarios, using up to eight robots and different shapes (e.g. circular and rectangular robots).

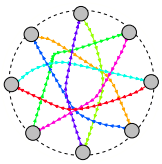
Based on the problem definition (see Section 1.2), we investigate the following questions.

1. What is the correct epsilon ε for the error-bound of the localization uncertainty?

2. Which VO type (VO, HRVO, RVO) performs best?

3. How do the methods compare when using different shapes and numbers of differential drive or holonomic robots?

[1] The pictures in the topright and topleft corner of each page are a flipbook of a simulation run with six and eight robots and ground truth. It is best viewed when printed double-sided or on screen, when the viewer is set to a two page layout.



4. How well do the methods perform in the presence of uncontrolled robots, i.e. dynamic and static obstacles?

The first two questions can be considered as parameter tuning for the proposed algorithms. With the third question we examine which combination of CALU or COCALU with the three velocity selection methods presented in Section 2.3 performs best for different kinds of robots. With the last question, we explore the actual real-life performance for different scenarios with static obstacles and dynamic obstacles. The dynamic obstacles are uncontrolled robots that do not take the movement of other robots into account.

6.1.1 System

All experiments in simulation are run on a single machine with a quad core 2.6 GHz Intel i7 processor and 6 GB of memory. Each setting is repeated 50 times and the results are averaged. Runs in which collisions occurred or which exceeded a time limit of 60 seconds are excluded from the averages. The completed runs are split into seven bins, and the variance of the batch means is used to calculate 90% confidence intervals using the students t-distribution with six degrees of freedom and $\alpha = 0.1$. The simulation are run in real time, since the message passing is an essential component of the described approach. ROS message passing uses real time serialization and deserialization, therefore increasing the simulation speed would lead to inaccurate results.

In the real-word setting, we have investigated the performance of CALU and COCALU using up to four differential drive turtlebots[2]. The robots are based on the iRobot's Roomba platform and have a diameter of 33.5 cm. In addition to the usual sensors, they are equipped with a Hokuyo URG laser-range finder to enable better localization in large spaces. All computation is performed on-board on a Intel i3 380UM 1.3 GHz dual core CPU notebook. Communication between the robots is realized via a 2.4 GHz WiFi link using a TCP/IP connection using a ROS topic shared between the different robots. Before the start the robots are driven remotely to their initial positions and AMCL is initialized with an approximated initial guess.

6.1.2 Common scenario

A common scenario for simulation and real life are a different number of robots located on a circle (equally spaced). The goals are located on the antipodal positions, i.e. each robot's shortest path is through the center of the circle (see [29, 1]). We use a circle with a radius of 1.7 meter in simulation and a radius of 1.4 meter in real life. The goal is assumed to be reached when the robots center is within a 0.15 meter radius of the true goal. As soon as a robot reaches its goal, it stops moving, i.e. it does not avoid other robots anymore or moves out of the way.

[2]For more information see: `http://turtlebot.com`.

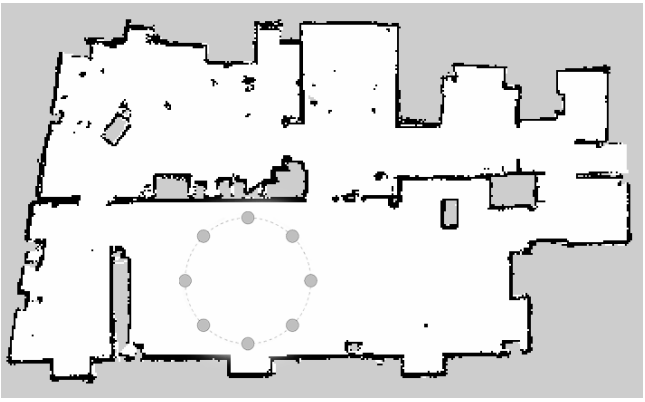


Figure 6.1: Static map used in all experiments for AMCL's laser scan matcher in simulation and real life. In the simulator stage, a world corresponding to the map was created. The robots initial positions and goals for the common scenario are located on the circle.

Figure 6.1 shows a sample configuration with eight robots. The map shows walls and static obstacles in our robotics laboratory. The map is created using gmapping[3] (for technical details see [16]) with a laser range finder mounted on top of one of the robots used for the real world experiments. For the simulator Stage, a world corresponding to the map was created.

The scenario can be tested with two different settings in simulation:

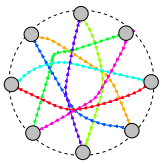
- *Ground Truth (GT)*: Each robot gets perfect position and velocity information through the simulation environment.
- *AMCL*: Each robot starts its own AMCL initialized with the exact pose corrupted by gaussian noise in x and y direction with $\sigma = 0.15m$. The AMCL particle cloud is then initialized around the given pose with a variance of $\sigma^2 = 0.15$ in x and y direction and with $\sigma^2 = 0.04$ in angular direction.

The robots used in simulation match the turtlebots used in the real world experiments. Thus we simulate a round differential drive robot with a radius of 0.17 meters for the simulation experiments. As a sensor source, a laser range finder is simulated, where the readings are corrupted with 5% gaussian noise. The exact settings and the deviations from the above are mentioned in the description for each experiments.

6.1.3 Performance measures

We measure several performance measures: a) number of collisions, b) time to complete a single run, c) distance travelled and d) jerk cost. The jerk cost

[3]For more information see: `http://www.ros.org/wiki/gmapping`



measures the smoothness of a path and is defined as:

$$Jerk_{lin} = \frac{1}{2} \int \dddot{\mathbf{x}}(t)\ dt \tag{6.1}$$

$$Jerk_{ang} = \frac{1}{2} \int \dddot{\theta}(t)\ dt \tag{6.2}$$

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.

6.2 Parameter tuning

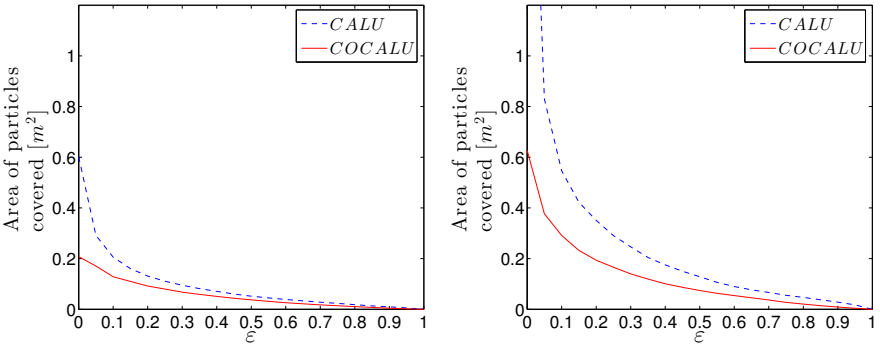
In this section, we investigate on the best parameters for the error-bound for the localization uncertainty (ε). The area of the approximation of the localization uncertainty for and the performance for the common scenario will be evaluated for various values of the error-bound ε. Additionally, we compare the performance of the different VO types for ClearPath and sampling based velocity selection.

6.2.1 Choosing the right error-bound for the localization uncertainty

As a first test, we compare the area of particles covered by CALU and COCALU for various values of ε. More specifically, we use the same AMCL particle cloud to calculate the CALU error distance ($d_{\mathbf{S}^*}$) as defined in Equation (3.9) for various error-bounds $\varepsilon \in [0, 1.0]$. For COCALU, we use the convex-hull peeling method until we hit our error-bound as described in Section 4.4. The area defined by the resulting convex hull is calculated and compared to the area of the CALU circle. The AMCL point clouds are generated by a robot driving ten times up and down, driving straight for four meters and then turning 180 degrees and returning to the starting place. This is done once in simulation and once in real-life. The sizes are calculated at each controller step, thus with 10 Hz. The results for each epsilon are averaged.

Secondly, we compare the average localization error (defined as the difference between the estimated pose and the ground truth pose) in simulation to the average distance of the points of COCALU convex hull to the CALU error distance ($d_{\mathbf{S}^*}$). We test this once with no sensor noise and once with 5% gaussian noise added to the sensor values. Again a robot is driving ten times up and down the simulated lab. We use the batch means method with 10 bins to calculate a 95% confidence interval for the average distances for CALU and COCALU.

As a third evaluation, we use the scenario defined in Section 6.1.2 with eight round differential drive robots in simulation. The laser scan is disrupted by 5% gaussian noise. The error-bound is varied from 0.0 to 1.0 increased in steps of 0.1. Each velocity selection method is evaluated 50 times and the results are averaged for CALU and COCALU respectively.



(a) Simulation run (b) Real life

Figure 6.2: Comparing the effect on the area of particles covered for different error-bounds (ε) in CALU and COCALU. (a) Resulting area covered in simulation. (b) Resulting area covered using a real robot.

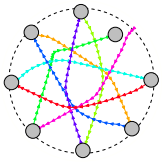
Results and Discussion

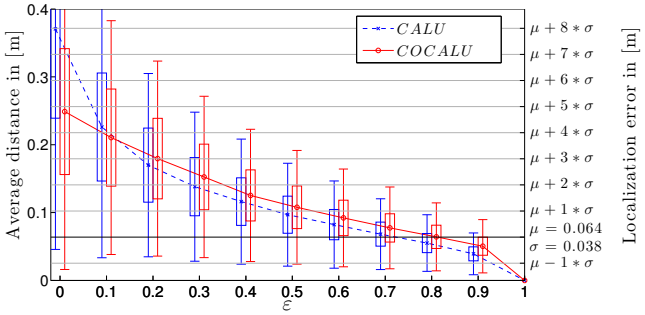
Figure 6.2 shows the effect of ε on the area of particles covered. We can see that the general trend is the same for simulation (Figure 6.2(a)) and real life (Figure 6.2(b)). COCALU generally covers a smaller area than CALU, which is to be expected. There can be cases in which COCALU generates a larger area than CALU, i.e. when there are various points with a high weight that are far away form the current position estimate. These points are included in the convex hull, as otherwise the weighted sum drops below the limit. In CALU the points may be excluded, since the particles closer to the current position estimate are taken first. However, as can be seen from the results this rarely happens.

From this result we can see, that error bound is necessary. Especially in real life with more uncertainties, the increase area of particles covered by CALU and COCALU is above $1m^2$ and $0.6m^2$ respectively. Thus, the error bound ensures that the virtual size of the robot is not increased so much that it is rendered immobile.

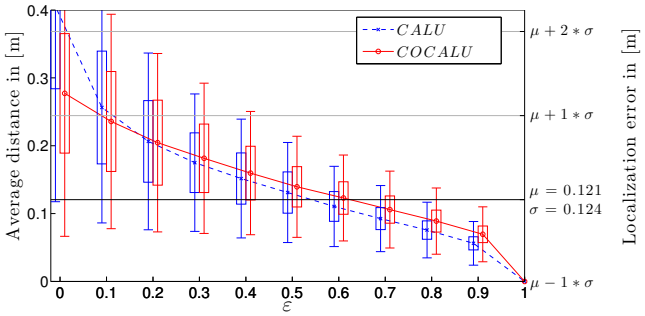
Figure 6.3 compares the localization error to the average distance to the outline of the calculated approximation of the localization uncertainty. We can see that the distance for CALU is slightly smaller on average starting from $\varepsilon = 0.2$. As expected, with no sensor noise the mean localization error and the standard deviation are very small with $\mu = 0.064\ m$ and $\sigma = 0.038\ m$. This increases to $\mu = 0.121\ m$ and $\sigma = 0.124\ m$, when disrupting the sensor with 5% gaussian noise.

The mean localization error is well below the confidence intervals of the average distance for $\varepsilon \leq 0.5$ with no noise and for $\varepsilon \leq 0.3$ with noise. When comparing only the means, we can see that for $\varepsilon \leq 0.7$ the mean average distance is higher than the localization error. With noise this decreases to $\varepsilon \leq 0.5$.





(a) Simulation run, no sensor noise

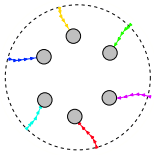


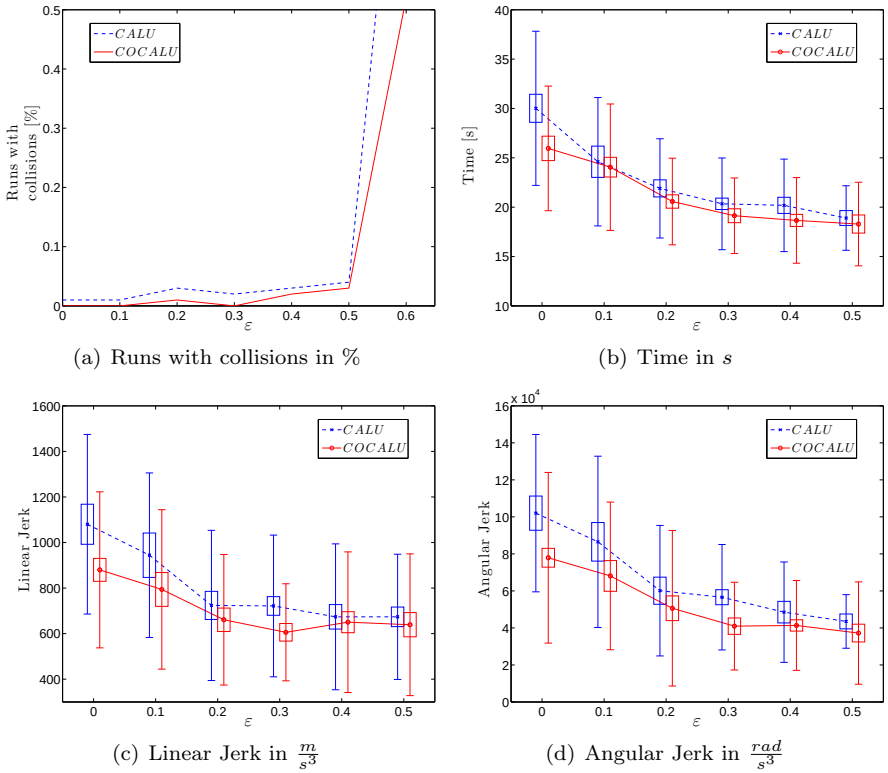
(b) Simulation run, sensor disrupted with 5% gaussian noise

Figure 6.3: Comparing the localization error in simulation with the average distance of the particle cloud covered for CALU and COCALU. For CALU this is the calculated radius and for COCALU we take the average distance from all points of the convex hull to the estimated pose. The plots are slightly offset to enhance comparability. The boxes define the 95% confidence interval for the mean and the bars show the standard deviation. (a) Results with no sensor noise. (b) Results for with the sensor data disrupted by 5% gaussian noise.

Thus, we expect that for $\varepsilon \gg 0.5$ many collision will occur, since the estimated localization uncertainty will not cover the actual localization error anymore.

The higher average distance for COCALU when compared to CALU for $\varepsilon > 0.1$ is due to the different approximation approaches. CALU adds particles with increasing distance, and as soon as the error bound is reached, the particle which is furthest away defines the radius of the approximation. COCALU removes convex hulls from the particle cloud by the peeling method. As soon as the boundary is reached, the current convex hull defines the area of the approximation. This can include points that are further away than the points included in CALU, and therefore it has a higher average distance, even though the total area covered is smaller for COCALU as seen in the previous experiment. With $\varepsilon = 0$ all particles are included in both approaches. Hence, COCALU has a lower average distance, since the convex approximation of the cloud is always





(a) Runs with collisions in %

(b) Time in s

(c) Linear Jerk in $\frac{m}{s^3}$

(d) Angular Jerk in $\frac{rad}{s^3}$

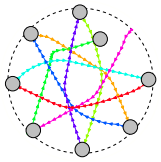
Figure 6.4: Comparing different error-bounds (ε) for CALU and COCALU. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. (a) Runs in which collisions occurred. (b) Average time needed to complete the run. (c) Linear Jerk. (d) Angular Jerk.

smaller than the circumscribed radius.

The results for the scenario with eight robots aligned on a circle are shown in Figure 6.4. When looking at the percentage of runs in which a collision occurred (Figure 6.4(a)), we can see that starting from $\varepsilon \geq 0.4$ many collisions occur. For $\varepsilon \geq 0.6$ in all runs for CALU and more than half for COCALU occurred collisions, thus for the further evaluation only runs from $\varepsilon \in [0, 0.5]$ were included.

When comparing the time needed (Figure 6.4(b)), it can be seen that the runtime decreases wit h an increase in the error-bound. CALU generally needs more time than the COCALU methods, though that is not always statistically significant. A similar trend can be seen when comparing the linear and angular jerk in Figure 6.4(c) and Figure 6.4(d) respectively.

Generally speaking, the standard deviation for the COCALU runs decreases



for $\varepsilon \leq 0.3$ and then increases again. For CALU it decreases until the last ε compared.

The observed results are expected due to the following; a larger ε leads to a smaller area of particles covered, thus also a smaller virtual footprint of the robots. This leaves more space to maneuver and hence the shorter times and smaller jerk costs. However, for $\varepsilon \geq 0.4$ a significant increase of collisions occurred. This is illustrated in Figure 6.3(b). The mean localization error is well below the average distance for $\varepsilon \leq 0.4$. This means that the areas before that value covers enough of the localization uncertainty to stay clear of the other robots. With $\varepsilon \gg 0.4$ the localization uncertainty is not covered enough, and thus the collisions will occur.

To summarize, we propose to use an error-bound $\varepsilon \in [0.2, 0.4]$ for this setup. However, for other robots in other domains the experiments have to be executed again to ensure the correct values. For the remainder of the experiments, we set $\varepsilon = 0.3$.

6.2.2 Comparing the different VO types

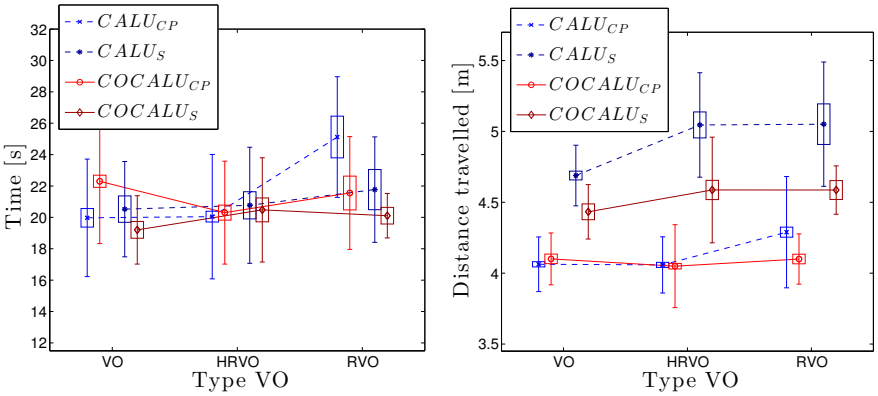
In Chapter 2, we have introduced three types of velocity obstacles - VO, RVO and HRVO. To evaluate these, we have chosen the common scenario defined in Section 6.1.2 with eight round differential drive robots in simulation. Each scenario is run 50 times, once with ground truth and once with AMCL. For AMCL runs, the laser scan is disrupted by 5% gaussian noise. The error-bound ε is set to 0.3 as explained in the previous section. We compare CALU and COCALU with the two velocity selection methods ClearPath and the sampling based approach, since ORCA is only defined for VOs.

As terminology we introduce $COCALU_S$, $COCALU_{CP}$ and $COCALU_{ORCA}$ for the three velocity selection methods, sampling based, ClearPath and ORCA, for COCALU respectively. The same naming convention is used for CALU. We will focus on the results for the runs running with AMCL, since these correspond to a real-world setting.

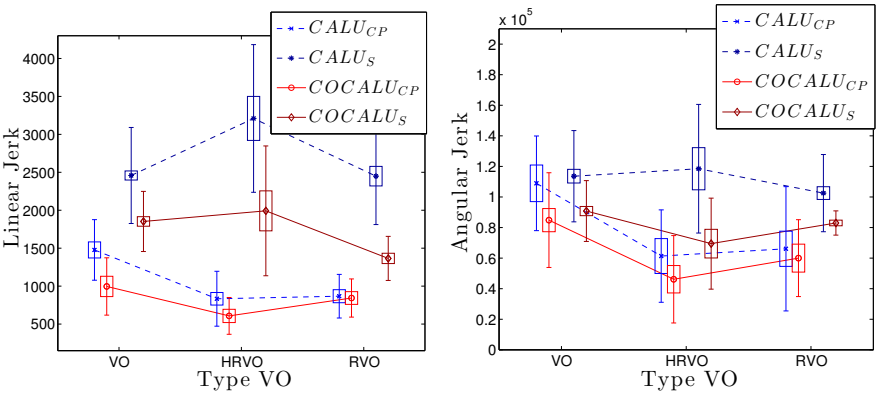
Results and Discussion

Figure 6.5 shows the results for using AMCL (for ground truth results, see appendix Figure A.1). The order from VO, to HRVO to RVO is chosen, since when revisiting Section 2.2, VO is the most restrictive, followed by HRVO and RVO is the least restrictive when comparing the area covered in velocity space. The number of runs with collisions are not shown, since in the AMCL runs only in the setting with HRVO and $COCALU_S$ a single collision occurred. The results for ground truth are generally faster and lower in distance and jerk (for better comparison, they are plotted in the same scale), but show similar trends as the AMCL results.

Looking at Figure 6.5(a) we observe that for $COCALU_{CP}$ the time needed rises with for RVO and VO when compared to HRVO; with $CALU_{CP}$ the times for HRVO and VO are similar but RVO is performing the worst. The sampling



(a) Time in s (b) Distance in m

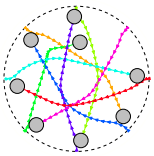


(c) Linear Jerk in $\frac{m}{s^3}$ (d) Angular Jerk in $\frac{rad}{s^3}$

Figure 6.5: Comparing the different VO types for CALU and COCALU with AMCL. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. (a) Average time needed to complete the run. (b) Distance travelled. (c) Linear jerk. (d) Angular jerk.

based approach is almost stable with a slight tendency that the VO is performing best. When comparing the distances travelled shown in Figure 6.5(b), similar trends can be detected, but for the sampling approach the tendency that VO is performing best is more visible.

When comparing the linear and angular jerk costs in Figure 6.5(c) and Figure 6.5(c) respectively, we can see that the paths are less smooth when using VOs together with ClearPath when comparing to RVOs and HRVOs. This is visible for $CALU_{CP}$ and as well for $COCALU_{CP}$. Using the sampling based approach the trends are different. For the linear jerk, HRVO shows the largest costs and RVO and VO perform similarly, while $COCALU_S$ with RVO is per-



forming best. For the angular jerk, the trend for $CALU_S$ is the same as for the linear jerk, but when using $COCALU_S$ with HRVO uses the lowest angular jerk, but with a high standard deviation.

It may seem surprising at first that the results differ so much for ClearPath and the sampling based approach. When revisiting the definitions of ClearPath and the sampling based approach, we see that ClearPath solves for the optimal new velocity, assuming that all the other robots do the same. Hence, we can speak of reciprocal collision avoidance. Therefore, with ClearPath, HRVO outperforms RVO and VO. RVO performs worse than HRVO due to the higher risk of oscillations coming from a symmetric setup. Hence, we can detect a "V" shaped trend, when looking at the runs with ClearPath. This is most visible in the angular jerk. However, when using the sampling based approach, we might not take the optimal new velocity, but a close to optimal collision free velocity. This might be diverted to any direction, hence the assumptions for the RVO and the HRVO do not hold anymore. Therefore, it is best to assume that the other agent is merely a dynamic obstacle, instead of an agent that takes responsibility in part of the collision avoidance.

To sum up, we propose to use the standard velocity obstacle (VO) for the sampling based approaches and the hybrid reciprocal velocity obstacle (HRVO) for ClearPath.

For the remainder of the experiments, we use ClearPath with HRVOs and the sampling based approach with VOs.

6.3 Comparison of different numbers and types of robots

In this section we will extensively examine the common scenario presented in Section 6.1.2. We will use two to eight simulated turtlebots, i.e. differential drive, round robots, and assessing all the combinations of the velocity selection methods with CALU and COCALU. Each setting will be run 50 times for ground truth and AMCL. During the AMCL runs, the sensor data is disrupted by 5% gaussian noise. The error-bound ε is set to 0.3, and the velocity obstacle type used for the runs with ClearPath is the HRVO and for the sampling based approach it is the standard VO, as explained and evaluated in the previous sections. This leads to twelve different settings, i.e. three COCALU settings, three CALU settings and each tested with ground truth and AMCL.

As a second type of robot we introduce rectangular shaped robots with holonomic drive. The width and length of the robots are 0.40 meters by 0.60 m respectively. The same twelve settings will be evaluated for this type of robots. The angular speed for the robots is defined such that the robots will preferably look into the direction in which they are driving. This is chosen, since the sensor source is usually located in front, thus we want to prevent driving sideways or backwards, since we have less sensor data in these directions. However, the turning is only allowed when the distance to the closest robot or obstacle is

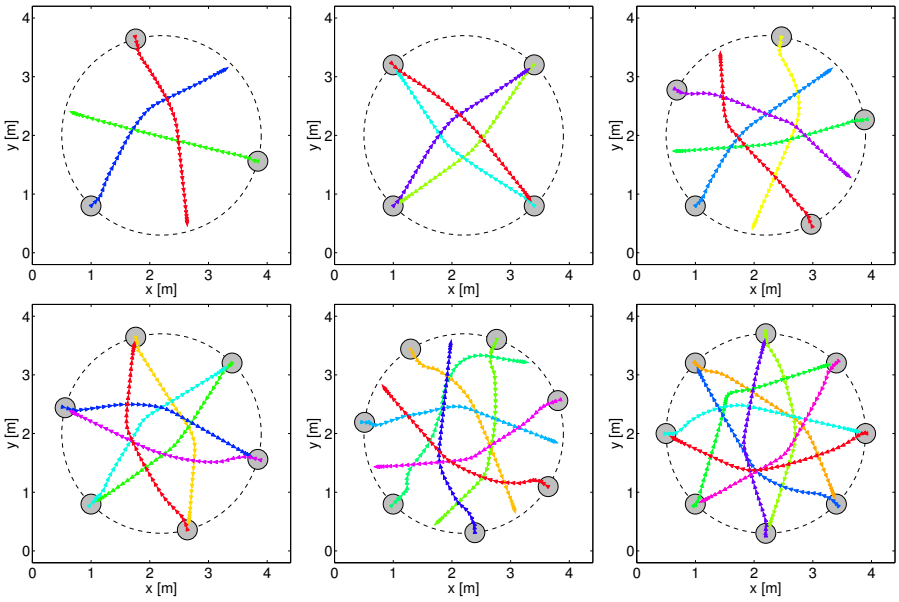


Figure 6.6: The smoothest robot trajectories observed with different numbers of circular robots using $COCALU_{CP}$ and ground truth. In the first row the results for 3, 4 and 5 robots are shown and in the second row, 6, 7 and 8 robots from left to right are presented.

large enough such that the turning will be collision free.

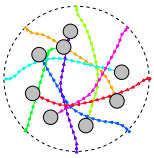
Real world experiments are performed with four turtlebots. Only the six settings using AMCL can be tested, since no external positioning method, e.g. a motion capturing system or an overhead camera, is available. Thus, we have to rely on the poses determined by the robots' AMCL. Furthermore, the real world runs take much more time and have to be supervised at all times. Therefore, these runs are only repeated ten times, to get an indication of the performance.

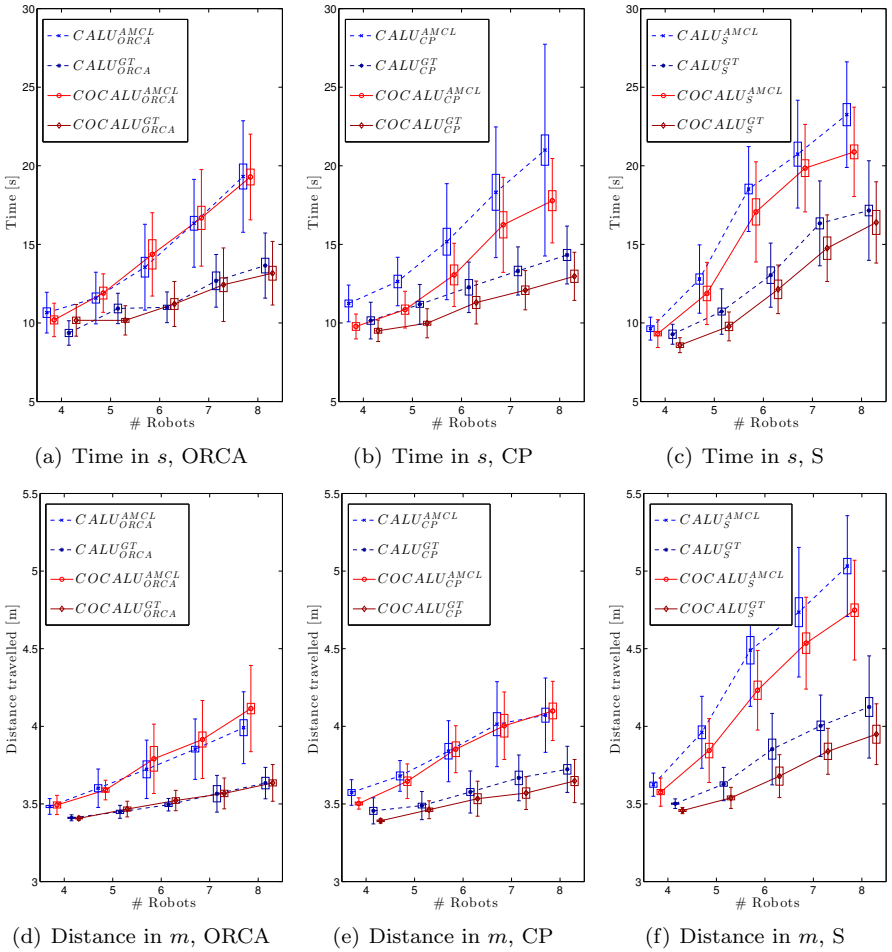
As a second real world setting, we used carton boxes to alter the shape of the turtlebots. The boxes are square with a length of 0.42 m for each side. Thus the six settings were also tested on quadratic shaped, differential drive turtlebots.

In Section 6.2.2 we introduced the notion of $COCALU_{CP}$ for the various combinations of CALU and COCALU with the three velocity selection methods. Additionally, we will use $COCALU_{CP}^{GT}$ and $COCALU_{CP}^{AMCL}$ to define when using ground truth (GT) and AMCL.

6.3.1 Results and discussion for round differential drive robots

Figure 6.6 shows smooth trajectories for three to eight robots. These are obtained by using ground truth positioning and $COCALU_{CP}$. When examining



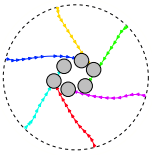


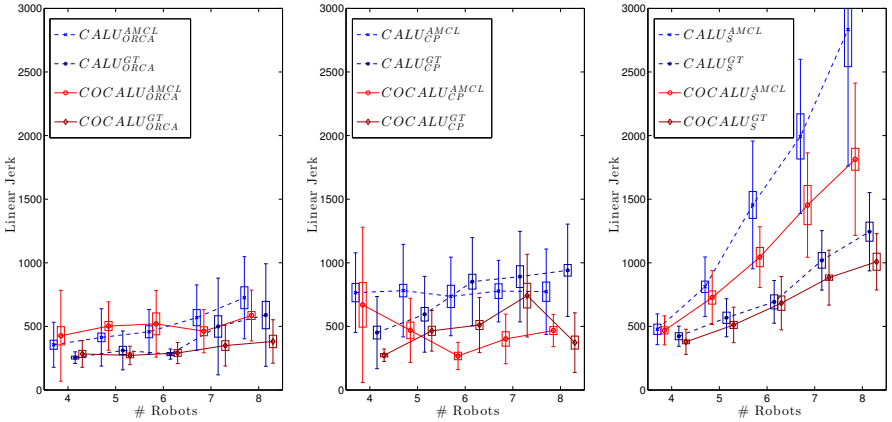
(a) Time in s, ORCA (b) Time in s, CP (c) Time in s, S

(d) Distance in m, ORCA (e) Distance in m, CP (f) Distance in m, S

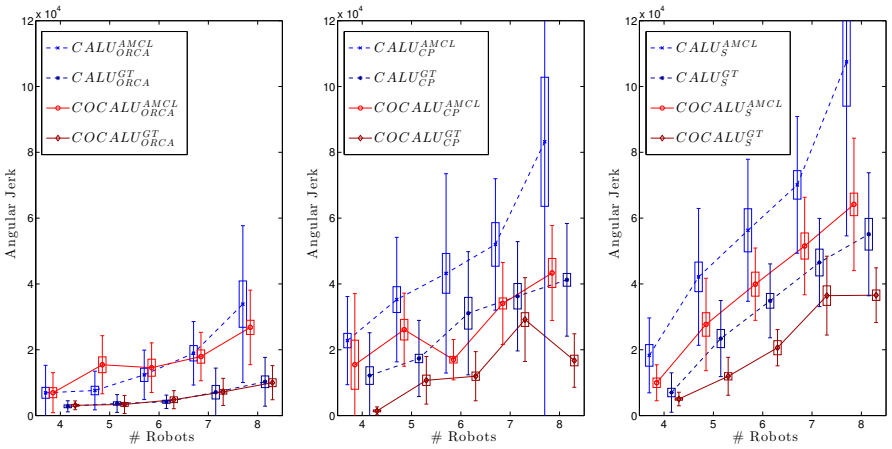
Figure 6.7: Comparing the time and distance with different number of round robots for CALU and COCALU. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. The first row compares the time needed with CALU and COCALU combined with the three velocity selection methods and second row compares the distance traveled.

the trajectories, it can be seen that the robots during the smoothest trajectories are forming a spiral to get to the other side of the circle. This behavior is not pre-programmed and emerges using the proposed algorithms. However, this only occurs, when all the robots located exactly symmetric and start almost at the same time. Even when using ground truth this is not always the case, since





(a) Linear Jerk in $\frac{m}{s^3}$, ORCA (b) Linear Jerk in $\frac{m}{s^3}$, CP (c) Linear Jerk in $\frac{m}{s^3}$, S



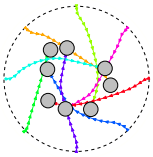
(d) Angular Jerk in $\frac{rad}{s^3}$, ORCA (e) Angular Jerk in $\frac{rad}{s^3}$, CP (f) Angular Jerk in $\frac{rad}{s^3}$, S

Figure 6.8: Comparing the jerk with different number of round robots for CALU and COCALU. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. The first row compares the linear jerk with CALU and COCALU combined with the three velocity selection methods and second row compares the angular jerk.

the message passing can lead to delays in communication.[4]

Only when using the sampling based approach with CALU and COCALU a single run had collisions, when using seven and eight robots. All runs without collisions were completed within the time limit of 60 seconds.

[4]The flipbook in the topright shows the run for six robots. On the top left, the run with 8 robots is presented. Both runs are with $COCALU_{CP}$ and ground truth.



To compare the performance of the six presented settings in ground truth and AMCL, Figure 6.7 and Figure 6.8 exhibit the run time, distance travelled and the jerk costs for all of the settings. Each row compares one performance measure for the three velocity selection methods, ORCA, ClearPath and sampling based from left to right. Each plot shows the results for using COCALU or CALU with AMCL or ground truth. The results for two and three robots are not shown, since for each setting, the results where not statistically different, hence they do not add value for the comparison.

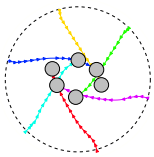
When comparing the time needed using ORCA (Figure 6.7(a)), it can be seen that CALU and COCALU almost perform equally well. There is no statistically significant difference between the two approaches. However, when looking at ClearPath and the sampling based approach in Figure 6.7(b) and Figure 6.7(c) respectively, it can be seen that COCALU generally performs better that CALU. The method using the least time is $COCALU_{CP}$, especially when looking at higher numbers of robots. Sampling works well for a lower number of robots, i.e. up to five robots, but after that the increase is very high. When comparing the distance travelled, similar trends can be seen for ORCA and the sampling based approach (Figure 6.7(d) and Figure 6.7(f)). However with ClearPath (Figure 6.7(e)), it is interesting that when using AMCL, the distance for CALU and COCALU are almost equal, even though COCALU takes less time to complete the run.

Looking at the results for the linear jerk using ORCA presented in Figure 6.8(a), we can see that all four methods are very close. Comparing it to using ClearPath (Figure 6.8(b)), we can see that CALU performs better using ORCA, while COCALU with AMCL performs best with ClearPath. Peculiar is the large variance when using four and five robots for $COCALU_{CP}$.

This is probably a simulation artifact, since there is not high variance for the time and distance. It could be the case that through the message passing delays and due to the fact that Stage does not take dynamic constraints into account, that the ground truth velocity jumps back and forth, from the maximum velocity to a smaller one. With six robots, the workspace is cluttered so much that the robots cannot drive at full speed at all, hence there are less jumps in the velocity.

Figure 6.8(c) shows the results for the sampling based approach. Overall, we can see that while COCALU generally outperforms CALU, the overall performance when comparing to ORCA and ClearPath is almost equal for four and five robots. For more than five robots, the sampling based approach generally outperforms by the other two approaches.

Very important for a smooth trajectory is the angular jerk, since changing the direction often, will lead to an increased angular jerk costs. Thus when comparing the three plots for ORCA, CP and sampling based, (Figure 6.8(d), Figure 6.8(e) and Figure 6.8(f)), we can see that ORCA performs best overall. CALU using ClearPath has a higher angular jerk cost than $COCALU_{CP}$, which explains the peculiarity which was observed before with CALU and COCALU using ClearPath, when comparing the times and distances travelled. While both travel roughly the same distances, COCALU requires less time to complete the runs. Hence, $CALU_{CP}$ uses more time maneuvering in place, which can be seen



# of robots	3	4	5	6	7	8
$CALU_{ORCA}$	0	0	0	0	0	0
$COCALU_{ORCA}$	0	0	0	0	0	1 (1)
$CALU_{CP}$	0	0	0	0	0	0
$COCALU_{CP}$	0	0	0	0	0	5 (2)
$CALU_{S}$	0	0	0	0	0	2 (1)
$COCALU_{S}$	0	0	0	0	3 (2)	11 (4)

Table 6.1: The number of collisions occurred during the simulation runs with rectangular robots and AMCL. The number in brackets denotes the number of runs in which the collisions occurred.

# of robots	3	4	5	6	7	8
$CALU_{ORCA}$	0	0	0	3	3	14
$COCALU_{ORCA}$	0	1	0	2	3	4
$CALU_{CP}$	0	0	0	3	8	14
$COCALU_{CP}$	0	0	0	0	2	2
$CALU_{S}$	0	0	0	7	21	26
$COCALU_{S}$	0	0	0	0	7	6

Table 6.2: The number of runs exceeding the time limit of 60 seconds during the simulation runs with rectangular robots and AMCL. The runs with collisions were already excluded from the runs.

from the angular jerk costs.

The sampling based approach has comparably low jerk costs for four and five robots, however afterwards, it increases rapidly. This trend can be observed in all performance measures. It is probably caused by the more cluttered environments. When there is enough space, there is a high probability that the preferred velocity or a sample close to it is outside of all velocity obstacles. Additionally, it is not optimally close, i.e. there is some additional safety area. However, in more cluttered environments, this is not the case anymore; then it might be that actually no sample is outside all velocity obstacles, and thus a velocity is chosen, which might lead to a collision at some point. In these cluttered environments, it is better to actually calculate the optimal velocity, instead of using the sampling based approach.

In summary, we conclude that with round robots, $COCALU_{CP}$ works best, while the best velocity selection method for CALU is ORCA. The sampling based approach works really well for up to five robots, but after that performance drops rapidly. Sample trajectories for the runs with $COCALU_{CP}$ and $CALU_{ORCA}$ can be found in the appendix Figure A.2.

6.3.2 Results and discussion for convex holonomic robots

When testing the algorithms with more complex shapes, i.e. rectangular holonomic robots, slightly more collisions occurred. The results are presented in

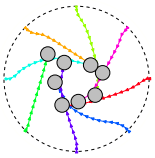


Table 6.1. It can be seen that CALU had less collisions than COCALU. This is due to the overestimation of the of the rectangular shape by the circumscribed radius. However, when looking at the runs that exceeded the time limit shown in Table 6.2, we can see that for CALU a lot more runs were not completed within 60 seconds, especially for seven and eight robots. These runs are usually a situation in which one or more robots are trapped in the center of the circle of robots that already reached their goal positions or when a robot is trapped between the wall and the other robots. Since CALU overestimates the localization uncertainty and the robots' footprints, this happens more often than with COCALU.

To further compare the performances of the proposed methods, Figure 6.9 compares the time and distance, and Figure 6.10 compares the jerk costs. When comparing the times to the previous runs with round robots, we can see that the averages are much higher for the rectangular robots. This also holds for the distances travelled and the jerk costs.

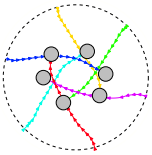
When considering the differences in time between the different velocity selection methods with CALU and COCALU, we can see that while for ORCA (Figure 6.9(a)) the time monotonously increases, with $COCALU_{CP}^{AMCL}$ there are two outliers, with five and with seven robots. For $COCALU_S$ it is interesting that the time increases also until seven robots and then for eight robots suddenly the time decreases again. In general, COCALU performs better than CALU. This is however not always statistically significant.

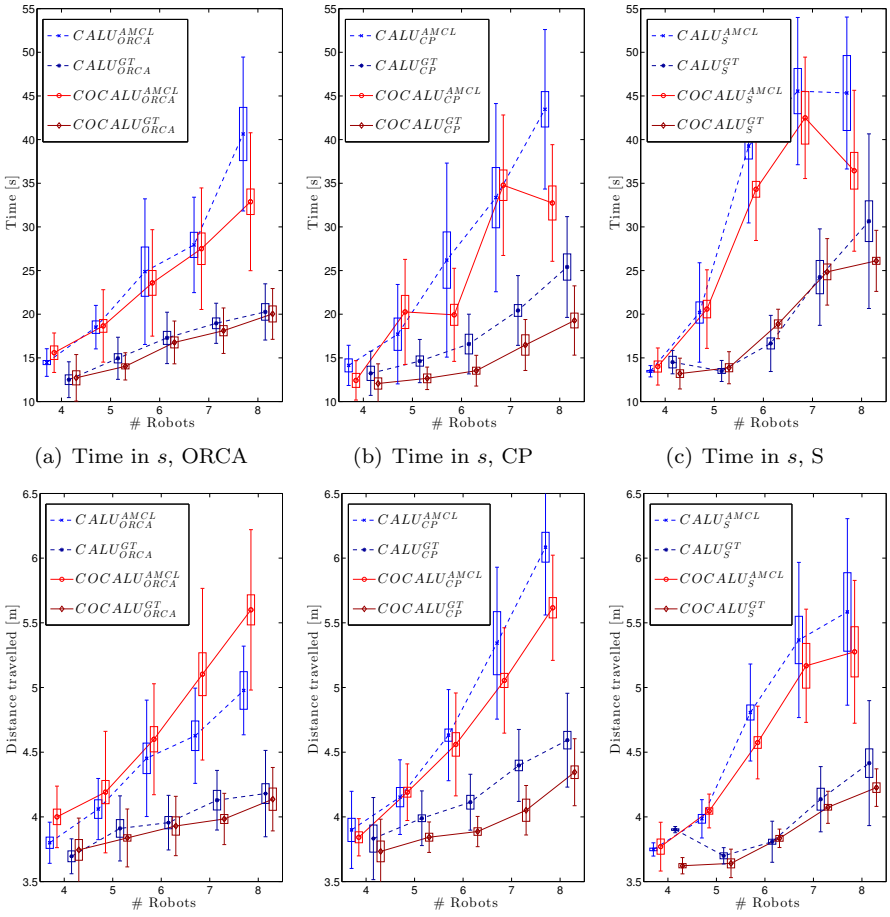
Comparing the distances traveled, we can see from Figure 6.9(d) that although $COCALU_{ORCA}^{AMCL}$ uses less time than $CALU_{ORCA}^{AMCL}$, the distance travelled using $COCALU_{ORCA}^{AMCL}$ is generally longer. In Figure 6.9(e), showing the the distance using ClearPath, we cannot see the outliers for five and seven robots for $COCALU_{CP}^{AMCL}$. Also for the sampling based approach, the drop in average time after seven robots cannot be detected in Figure 6.9(f).

Figure 6.10(a) and Figure 6.10(d) show the results for linear and angular jerk for $CALU_{ORCA}$ and $COCALU_{ORCA}$. Here we can see that $COCALU_{ORCA}^{AMCL}$ uses significantly more linear jerk than $CALU_{ORCA}^{AMCL}$ for most numbers of robots. That is probably also the reason for the longer path lengths detected before. The angular jerk for $COCALU_{ORCA}^{AMCL}$ and $CALU_{ORCA}^{AMCL}$ is quite low when compared to the other methods.

As already concluded with round robots $CALU_{CP}$ with AMCL does not work well, which can also be seen from Figure 6.10(b) and Figure 6.10(e). In both cases $CALU_{CP}$ performs significantly worse than the corresponding $COCALU_{CP}$ runs. The angular jerk for seven robots and $COCALU_{CP}^{AMCL}$ is above the trend and has a larger standard deviation than the runs with the other numbers of robots. This might explain the outlier for seven robots. For five robots, there is an indication that the standard deviation and mean are a bit higher as well, but this is not as visible as for seven robots.

There is again a peculiarity with four and five robots, when looking at the angular and especially the linear jerk with $COCALU_{AMCL}^{GT}$. The linear jerk for four and five robots is much higher then we would expect. This can be again explained by simulation artifacts as explained in the previous section.





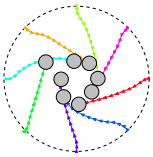
(a) Time in s, ORCA (b) Time in s, CP (c) Time in s, S

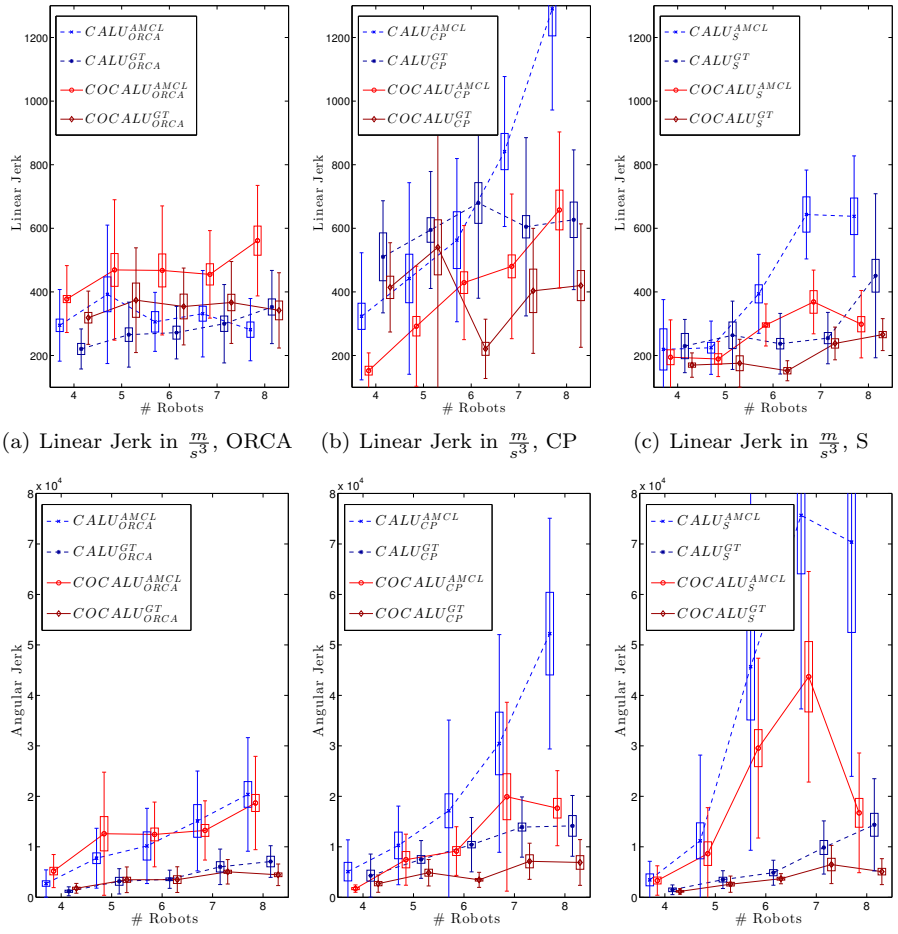
(d) Distance in m, ORCA (e) Distance in m, CP (f) Distance in m, S

Figure 6.9: Comparing the time and distance with different number of rectangular robots for CALU and COCALU. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. The first row compares the time needed with CALU and COCALU combined with the three velocity selection methods and second row compares the distance traveled.

The outliers with five and seven robots might be explained by the asymmetric setup when using an uneven number of robots. In these cases there is no head on collision from the start, thus the avoidance only occurs later. With three robots there is enough space, thus this is not much of a problem, but with five and especially with seven robots, we can seen that it takes much more effort.

When looking at the results for the sampling based approach, presented in Figure 6.10(c) and Figure 6.10(f), we can also see that $CALU_S$ also does



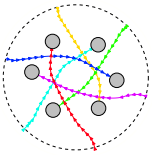


(a) Linear Jerk in $\frac{m}{s^3}$, ORCA (b) Linear Jerk in $\frac{m}{s^3}$, CP (c) Linear Jerk in $\frac{m}{s^3}$, S

(d) Angular Jerk in $\frac{rad}{s^3}$, ORCA (e) Angular Jerk in $\frac{rad}{s^3}$, CP (f) Angular Jerk in $\frac{rad}{s^3}$, S

Figure 6.10: Comparing the jerk with different number of rectangular robots for CALU and COCALU. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. The first row compares the linear jerk with CALU and COCALU combined with the three velocity selection methods and second row compares the angular jerk

not perform well, when compared to $COCALU_S$. Both, for angular and linear jerk the results are significantly higher when using more than five robots. $COCALU_S$ on the other hand, while performing really well when looking at the linear jerk, it performs also very badly for the angular jerk, when compared to the other velocity selection methods. The peculiar drop for eight robots using $COCALU_S^{AMCL}$ is visible.



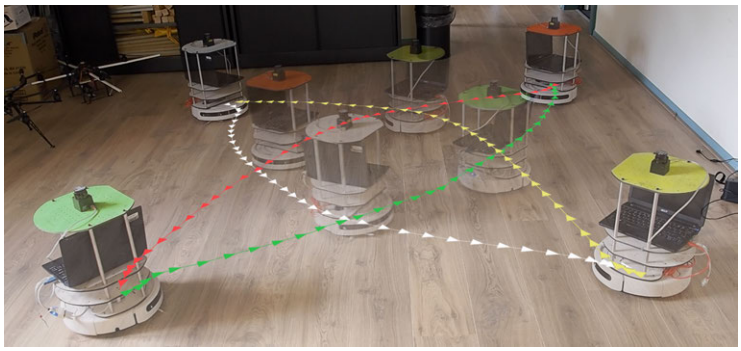


Figure 6.11: $COCALU_{CP}$ with four round turtlebots, the actually driven paths are overlaid on the picture. The pictures are two photos of the real run, one from the start and one during the middle of the run, blended together.

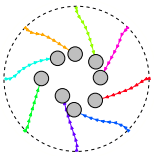
The drop for eight robots using $COCALU_S$ might again be a simulation artifact. However another explanation might be that because the workspace gets crowded that it is essentially not possible anymore to drive through the center, the sampled velocities that get selected are always outside of the circle. Thus the spiral effect emerges right away.

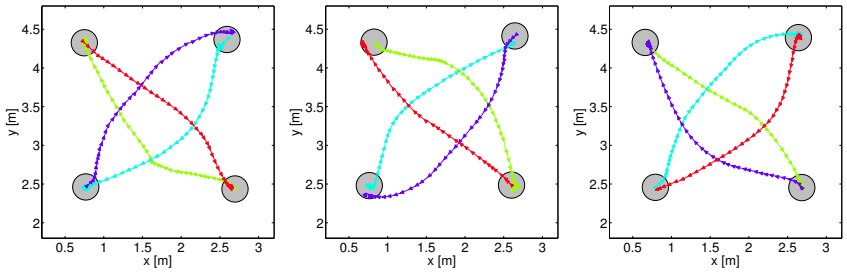
To conclude, we can say that for rectangular shaped holonomic robots, the results are similar to the results with round robots. While for CALU the best performing velocity selection method is again ORCA, for COCALU it is usually ClearPath. However in some of the scenarios $COCALU_{ORCA}$ or $COCALU_S$ work better than $COCALU_{CP}$. Sample trajectories for the runs with $COCALU_{CP}$ and $CALU_{ORCA}$ can be found in the appendix Figure A.3.

6.3.3 Real world results

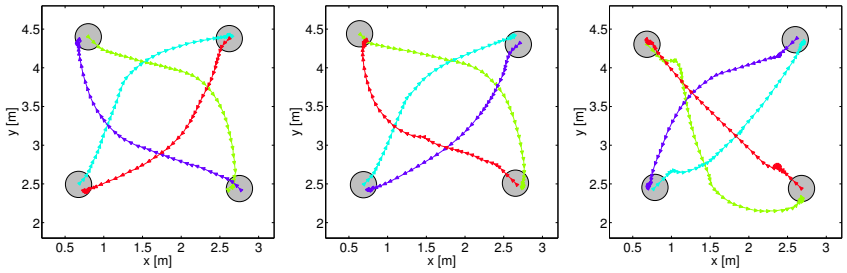
The common scenario was also tested in real world using four turtlebots. A sample run for $COCALU_{CP}$ is shown in Figure 6.11. The paths are created from following the poses reported by the AMCL running on each robot, since no external localization method is available. Thus the paths might be off by the localization error. The plotted trajectories are perspectively projected in order to match the angle of the camera.

The runs were repeated ten times and the averages and standard deviation are shown in Table B.1 in the appendix. Due to the low number of repetitions, the result are not statistically significant. However, the times and standard deviations across all the runs are pretty similar. This result is similar to the simulation runs. When revisiting the results for simulation, we see that for four robots the distinction between the various methods is not always statistically significant even with 50 runs. Figure 6.12 and Figure 6.13 shows three sample trajectories for the four turtlebots using the six settings. The trajectories for $COCALU_{ORCA}$, $CALU_{ORCA}$, $COCALU_{CP}$ and $CALU_{CP}$ are shown in the first figure and $COCALU_S$ and $CALU_S$ are exhibited in the latter figure.

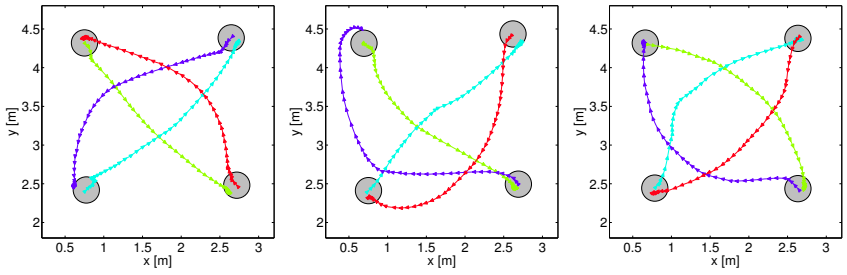




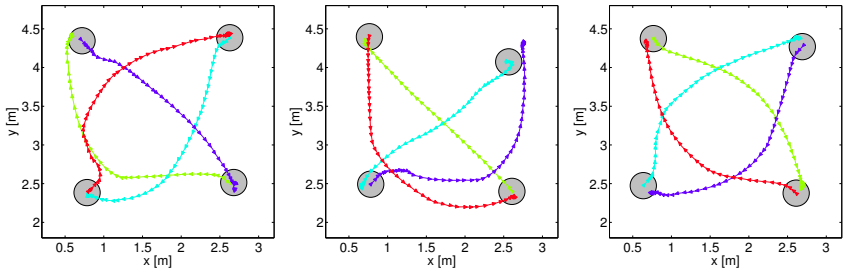
(a) $COCALU_{ORCA}$



(b) $CALU_{ORCA}$

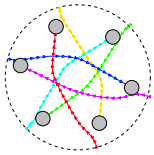


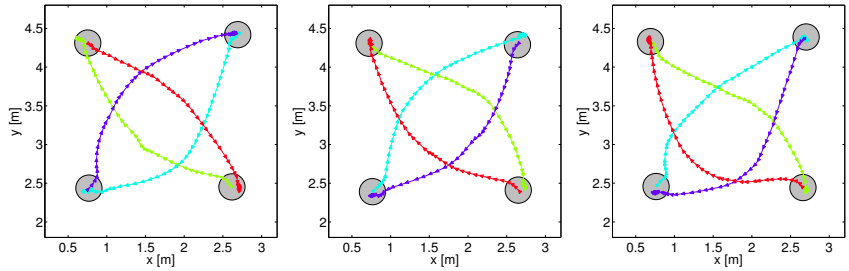
(c) $COCALU_{CP}$



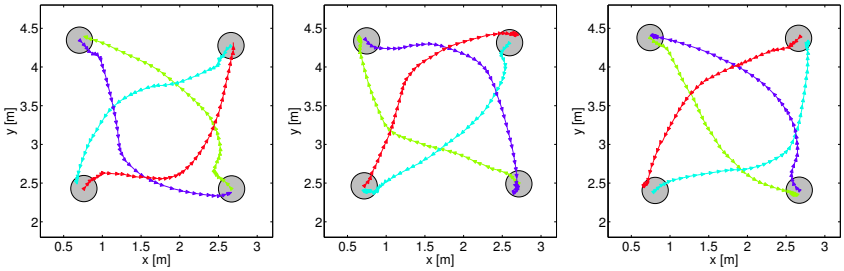
(d) $CALU_{CP}$

Figure 6.12: Sample trajectories with four turtlebots in real life. $COCALU_{ORCA}$ is in the first row, followed by $CALU_{ORCA}$, $COCALU_{CP}$ and $CALU_{CP}$.



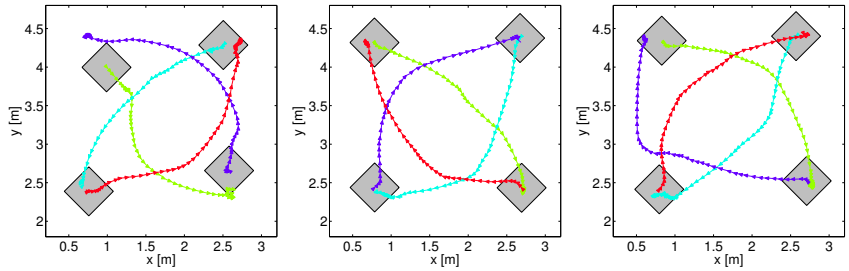


(a) $COCALU_S$

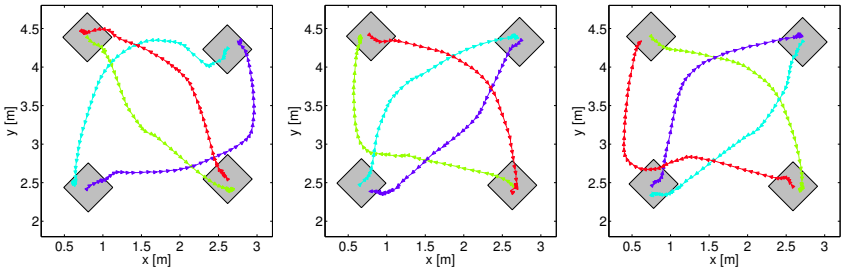


(b) $CALU_S$

Figure 6.13: Sample trajectories with four turtlebots in real life. $COCALU_S$ is in the first row, followed by $CALU_S$.

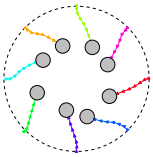


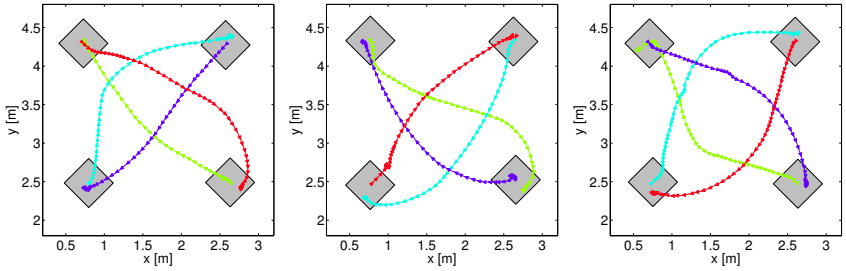
(a) $COCALU_S$



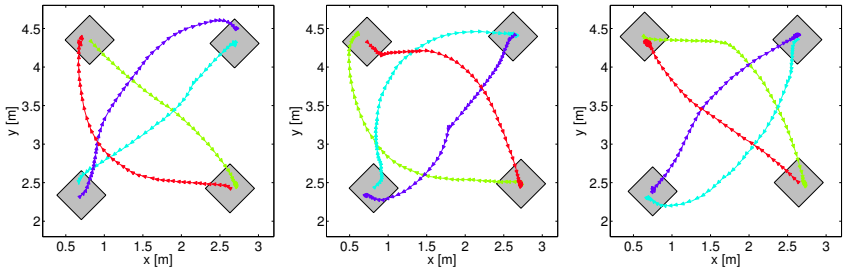
(b) $CALU_S$

Figure 6.14: Sample trajectories with four quadratic turtlebots in real life. $COCALU_S$ is in the first row, followed by $CALU_S$.

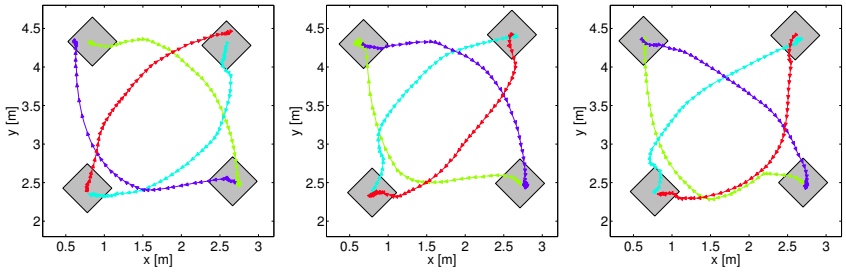




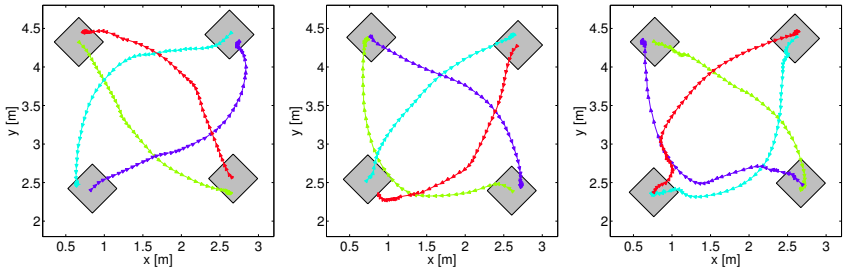
(a) $COCALU_{ORCA}$



(b) $CALU_{ORCA}$



(c) $COCALU_{CP}$



(d) $CALU_{CP}$

Figure 6.15: Sample trajectories with four quadratic turtlebots in real life. $COCALU_{ORCA}$ is in the first row, followed by $CALU_{ORCA}$, $COCALU_{CP}$ and $CALU_{CP}$.

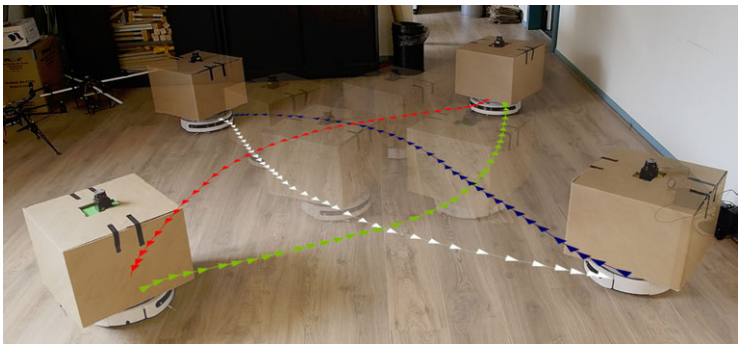


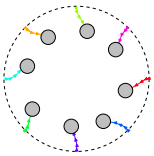
Figure 6.16: $COCALU_{CP}$ with four quadratic turtlebots, the actually driven paths are overlaid on the picture. The pictures are two photos of the real run, one from the start and one during the middle of the run, blended together.

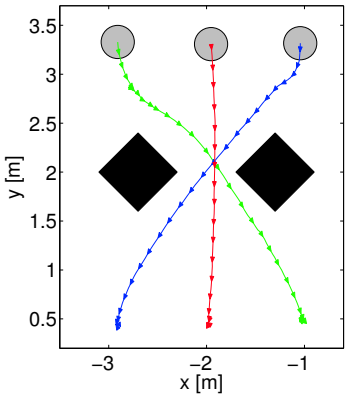
When comparing the different trajectories, it can be seen that they vary between different settings. However, the spiraling effect can be detected in many runs. Interestingly, using $COCALU_S$ and $CALU_S$, results in the spiral for every run, while using the optimal velocity selection methods results in more different trajectories. This is similar to the results in simulation, where the sampling based approach works well up to five robots.

For the second approach the shape of the turtlebots is altered by attaching a 0.42m times 0.42m box on the robots. The footprint in the configuration file are adapted accordingly, so that the robots know their different shape. These runs are also repeated ten times and the averages and standard deviation are shown in Table B.2 in the appendix to give an indication of the performance. Figure 6.3.3 shows a sample run using the four quadratic turtlebots when using $COCALU_{CP}$. The reported trajectories are again overlaid on top of two photos of the run.

A selection of three trajectories for each setting for the quadratic robots is presented in Figure 6.15 for the runs with $COCALU_{ORCA}$, $CALU_{ORCA}$, $COCALU_{CP}$ and $CALU_{CP}$ and in Figure 6.14 for $COCALU_S$ and $CALU_S$. Looking at the trajectories, the patterns are even more diverse than with the original round shaped turtlebots. However, we can still detect the spiraling effect in many runs.

In summary, all the presented approaches also work in the real world scenario. However these are merely a proof of concept, since due to the physical limitations of the real world setting, i.e. less repetitions possible, no ground truth positioning system, "only" four robots available, we have to rely on the simulation results to properly compare the performances of the proposed systems. The differences for CALU and COCALU for the three velocity selection methods are not statistically significant for the ten runs performed for each setting. We will examine the behavior in real life in further detail with more complex and different scenarios in the next section.





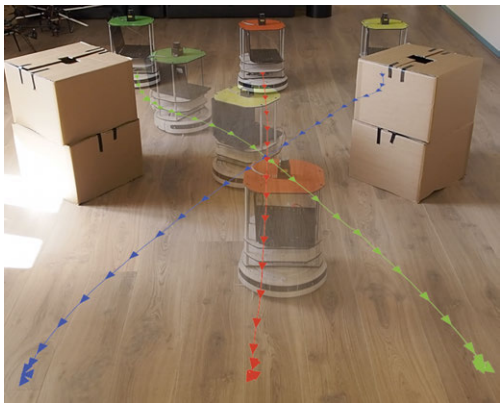


Figure 6.17: $COCALU_{CP}$ with three turtlebots and two static obstacles. All robots have to pass through a narrow passage, while the goal locations are located on the other side. The outer two robots have to switch position. The left picture shows the trajectories plotted in a graph together with the obstacles. On the right, the same trajectories are overlaid on a picture of the actual run.

6.4 Moving and static obstacles in real world scenarios

In this section examine the behavior of the on average best performing algorithm $COCALU_{CP}$ in more complex real word scenarios. We will introduce static obstacles that are not in the static map and moving obstacles, which are robots that do not take the other robots into account. Hence, these uncontrolled robots will drive straight towards the goal while avoiding only static obstacles. Thus, the other controlled robots have to take full responsibility of avoiding the uncontrolled robots.

The first scenario will be three robots using $COCALU_{CP}$ that have to pass through a narrow passage defined by two obstacles. They start aligned next to each other parallel to the opening of the passage and the goals are located at the other side of the passage while the the outer two robots have to switch position. Figure 6.17 shows the setup and one run.

The second scenario is with four robots, in which one is used as dynamic obstacle. Hence, it will drive straight towards the goal. Again, there is a narrow passage through which all robots have to pass. The three controlled robots are aligned on a line with the opening of the passage. The uncontrolled robot is on the same line on the other side of the passage. Figure 6.19 shows the setup and one run.

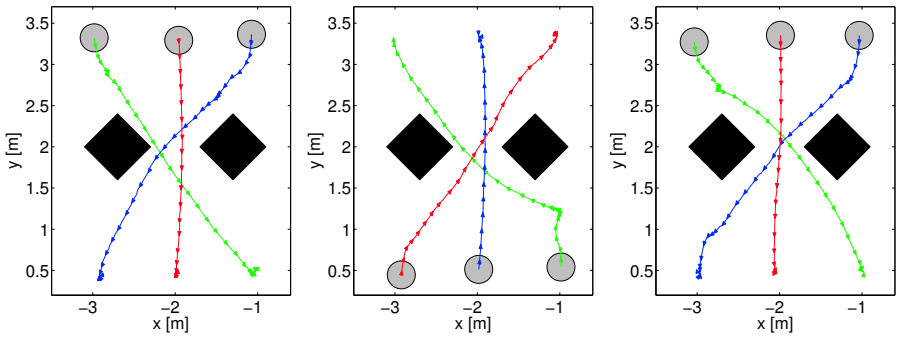


Figure 6.18: $COCALU_{CP}$ with three turtlebots and two static obstacles. All robots have to pass through a narrow passage, while the goal locations are located on the other side. The outer two robots have to switch position. The three pictures show plots of the following the poses reported by the robots' AMCL, since no external ground truth localization method is available.

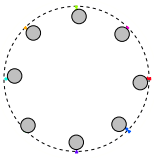
6.4.1 Results and discussion

Figure 6.17 shows a run for the first scenario. On the left hand side the trajectories are plotted together with the obstacles. The positions of the obstacles are measured in the real world and the obstacles are plotted at the measured positions. The trajectories are following the robots' localized poses. Hence it might be not completely accurate, since there is no external global positioning system like an overhead camera to provide an actual ground truth position. Nevertheless, we can see that the trajectories match the actually driven paths well that can be reconstructed by the pictures taken of the real run. On the right hand side, we exhibit a photo in which a picture from the starting position and from the middle of the run are overlaid together with the reported paths. The paths are perspectively transformed to match the angle of the camera.

The result shows that the robot in the top left position drives slowly in the beginning, which is represented by the arrows being closer together. The middle robot moves basically in a straight line through the passage. The rightmost robot passes through the passages shortly after the middle robot and before the left robot. This behavior emerged using only the described methods and using the goal position as preferred velocity and no negotiation between the robots. The only communication between the robots is the exchange of positions, shape and speeds.

Figure 6.18 exhibits three more runs of the defined scenario. In all cases, the middle robot is the first to pass through the passage. The left most robot is the last one to pass for the two runs going from top to down, shown on the left and right in the picture. In the middle, a run going from down upwards is shown. Here the robot starting in the down-right corner its the last to pass through the passage.

These runs show the natural variance in the approach. There are no runs



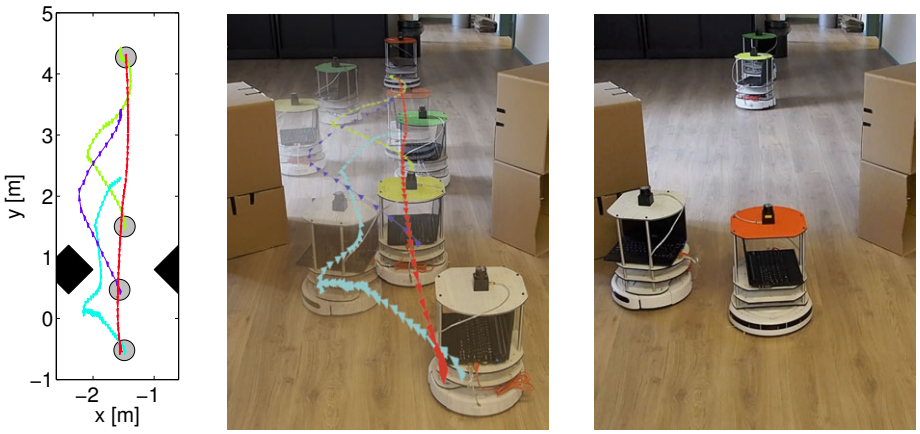
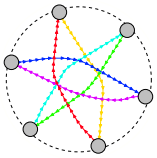


Figure 6.19: $COCALU_{CP}$ with three turtlebots and an uncontrolled moving obstacle. The three controlled robots are located together on one side and the uncontrolled robot is on the other side. Some robots still have to pass through the small passage defined by the two static obstacles. On the left the left a trajectory plot is shown of the following the poses reported by the robots' AMCL, since no external ground truth localization method is available. In the middle, the same trajectories are overlaid on a picture of the actual run. The right most picture shows the situation that the white robot waited until the red uncontrolled robot passed so that it is free to move to the passage.

which are the same, since there is always a small change in the environment. For instance, the localization might be a bit off and "jump" to a better location at some point. This can be seen on the right picture of Figure 6.18. The robot starting from the top left corner has a paths that is not continuous in the beginning. However, since $COCALU_{CP}$ takes the localization uncertainty into account, it still results in a collision free motion in which all robots reached their goals.

On run for the second scenario is shown in Figure 6.19. Again the left hand plot shows the trajectories and in the middle a picture of two photos (from the start and around the middle of the run) are blended together and overlaid with the perspectively projected paths. The three robots starting below are the robots that are controlled, i.e. they do take care of collision avoidance of other robots. The upmost robot is considered a dynamic obstacle, and thus moves directly towards is goal location, discarding the existence of the other robots. This means that the three controlled robots have to fully avoid the other robot. The "uncontrolled robot" still broadcasts its pose, velocity and shape information, together with the fact that it is an uncontrolled robot.

The plot on the left shows that the robot indeed drives directly towards its goal location on the other side of the passage. The second and third robot from below are actually located in or almost before the passage, thus their trajectories



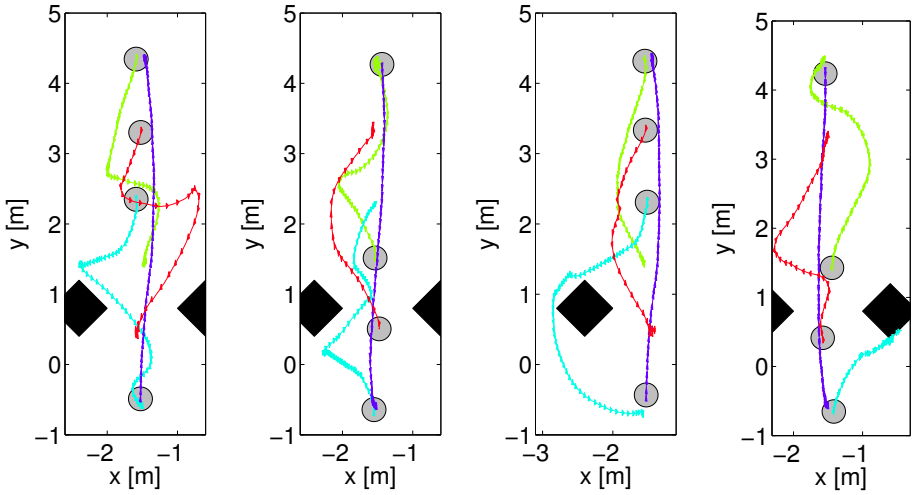
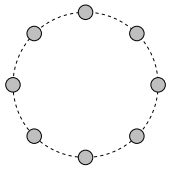


Figure 6.20: $COCALU_{CP}$ with three turtlebots and an uncontrolled moving obstacle. The three controlled robots are located together on one side and the uncontrolled robot is on the other side. Some robots still have to pass through the small passage defined by the two static obstacles. The four pictures show plots of the following the poses reported by the robots' AMCL, since no external ground truth localization method is available.

are basically only avoiding the robot coming from above. The robot starting at the lowest position has still to pass through the hallway and avoid the coming robot. Interestingly, it does so by first avoiding the coming uncontrolled robot, i.e. dynamic obstacle, and waiting just before the left static obstacle until the dynamic obstacle has passed and then the robot continues towards its goal. This is shown in Figure 6.19 on the right; the robot that has started lowest is still waiting for the uncontrolled robot to pass, before continuing towards its goal.

The trajectories of four additional runs are presented in Figure 6.20. These show that this task has more variance in the resulting trajectories. The second plot from the left exhibits quite a similar run to the one shown also in Figure 6.19. The first and third plots from the left similar scenarios, but reversed. Thus all robots have still to pass through the passage, except for the upmost robot, which has a goal location located just before the passage. On the third plot something unexpected has happened; the robot closest to the uncontrolled robot has not gone through the passage way, but around it. This has probably happened because it already has avoided the coming dynamic obstacle so far to the left that the projection of the preferred velocity is on the other side of the obstacle. Thus it moved around it. The forth plot shows a run in which a deadlock position occurred. The robot starting lowest avoids to the right and drives close to the obstacle. However, the preferred velocity points directly towards the goal, but there is the static obstacle. Hence the optimal velocity becomes



zero.

To conclude, it can be seen that the proposed system works well even in more complex scenarios with static and moving obstacles. However, some limitation were detected, since in some runs the robots got in a deadlock position before an obstacle. This is due to the fact that only a simple goal planner is used, where the preferred velocity points straight to the goal, i.e. when the robot is behind an obstacle and the goal is located on the other side, the projection of the preferred velocity on the velocity obstacle in velocity space might be at the origin. Thus the optimal velocity is then the zero velocity. Hence, the robot will not move anymore, since this situation does not change. These situations could be overcome by a smarter global planner that would re-plan in such situations to go around the static obstacle. But this is out of the scope of this thesis.

6.5 Summary

In summary, we have extensively evaluated the proposed algorithms. We have tested and investigated various different scenarios and settings. The parameter ε for the localization uncertainty has been tuned and the various velocity obstacle types has been compared for ClearPath and the sampling based approach together with CALU and COCALU.

Both proposed algorithms, are extensively compared with various numbers of robots in simulation and in real life. It has been shown that for CALU in most of the cases ORCA performs best and for COCALU the best velocity selection method is ClearPath. However in runs with small numbers of robots many of the results are not statistically significant, since the differences are not large enough and the number of completed runs was too low, i.e. in the real world setting. Furthermore, the sampling based approach showed promising results for low numbers of robots, but for more than five robots, the performance drops rapidly.

Finally, we have tested the on average best performing algorithm $COCALU_{CP}$ in more complex scenarios that included static and dynamic obstacles. These results show that the algorithm works well in many cases, but a combination with a smarter global planner would help to overcome some deadlock situation, in which the optimal velocity becomes zero. Overall it can be concluded that the proposed algorithms can be applied in various scenarios, i.e. including multiple controlled robots, static obstacles and uncontrolled robots.

Chapter 7

Conclusions

As low-cost mobile robots are becoming accessible, workspaces are likely to feature not one buy multiple robots.

The traditional approach uses a centralized planner that controls all the robots in the working space. However this is only feasible for a small number of robots and not robust. If the planner fails, the whole system breaks. This thesis combines decentralized approaches that show promising results in simulation with per-agent onboard localization to realize a multi-mobile robot navigation solution. The contributions of this research work can be summarized as follows.

We combine the use of per-agent base localizations with decentralized local collision avoidance algorithms based on the velocity obstacle paradigm. In particular, we propose a system that is using only limited local communication to share position, velocity and shape information, while no negotiation or path planning is involved during the collision avoidance.

Most importantly, this thesis introduces the means of an error-bound for the localization uncertainty. This can be used to balance the error introduced by localization and the virtual increase of the robots' footprints to allow safe navigation. We present a circular and a close convex approximation of the particle cloud of the localization algorithm that used in combination with the robots' footprint to calculate new collision free velocity commands close to the preferred velocity.

We present how static obstacles can be incorporated in this approach to provide a fully distributed local navigation solution. A laser scanner (LIDAR) as sensor source is assumed; however, a solution to deal with 3D point clouds is also provided.

The resulting approaches are evaluated and tested under various conditions and scenarios in simulation and real life. Parameters for the bound of the localization error are determined and the various types of velocity obstacles are compared. The two approaches to bound the localization uncertainty are further evaluated with three different velocity selection methods and various numbers of differently shaped robots. More complex scenarios including a small passage defined by two static obstacles and using one robot as uncontrolled dynamic

obstacle are investigated using the best performing algorithm.

Recommendations for future research include sensor based estimation of the relative positions and eventually of the velocity and shape information to eliminate the need of communication. A second extension can be done to incorporate human tracking into the system. The recent work of Mitzel et al. [23] show a promising approach that can be incorporated into the proposed system. Additionally, Berg et al. recently introduced the acceleration velocity obstacles (AVO) [31] that can be used instead of the linear velocity obstacle to enable more accurate incorporation of dynamic and movement constraints.

To conclude, even though *convex outline collision avoidance under localization uncertainty (COCALU)* shows promising results, there is no silver bullet that works well in all cases. Or as Nietzsche said: "You have your way. I have my way. As for the right way, the correct way, and the only way, it does not exist."

Bibliography

[1] J. Alonso-Mora, A. Breitenmoser, M. Rufli, P. Beardsley, and R. Siegwart. Optimal reciprocal collision avoidance for multiple non-holonomic robots. In *Proceedings of the 10th International Symposium on Distributed Autonomous Robotic Systems (DARS)*, 2010.

[2] D. Althoff, J. Kuffner, D. Wollherr, and M. Buss. Safety assessment of robot trajectories for navigation in uncertain and dynamic environments. *Autonomous Robots*, 32:285–302, 2012. 10.1007/s10514-011-9257-9.

[3] K. Azarm and G. Schmidt. Conflict-free motion of multiple mobile robots based on decentralized motion planning and negotiation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, volume 4, pages 3526 –3533 vol.4, apr 1997.

[4] J. Bruce and M. Veloso. Safe multirobot navigation within dynamics constraints. *Proceedings of the IEEE*, 94(7):1398 –1411, july 2006.

[5] T. M. Chan. Optimal output-sensitive convex hull algorithms in two and three dimensions. *Discrete & Computational Geometry*, 16:361–368, 1996.

[6] B. Chazelle. On the convex layers of a planar set. *IEEE Transactions on Information Theory*, 31(4):509–517, 1985.

[7] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen. CALU: Collision avoidance with localization uncertainty [Demonstration]. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, June 2012.

[8] D. Claes, D. Hennes, K. Tuyls, and W. Meeussen. Collision avoidance under bounded localization uncertainty. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Algarve, Portugal, October 2012.

[9] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press, 2001.

[10] A. Ferrara and M. Rubagotti. A dynamic obstacle avoidance strategy for a mobile robot based on sliding mode control. In *IEEE Control Applications (CCA) and Intelligent Control (ISIC)*, pages 1535 –1540, july 2009.

[11] P. Fiorini and Z. Shiller. Motion planning in dynamic environments using velocity obstacles. *International Journal of Robotics Research*, 17:760–772, July 1998.

[12] D. Fox. Kld-sampling: Adaptive particle filters. In *Advances in Neural Information Processing Systems 14*. MIT Press, 2001.

[13] D. Fox. Adapting the sample size in particle filters through kld-sampling. *International Journal of Robotics Research*, 22, 2003.

[14] D. Fox, W. Burgard, and S. Thrun. The dynamic window approach to collision avoidance. *Robotics Automation Magazine, IEEE*, 4(1):23 –33, mar 1997.

[15] B. P. Gerkey, R. T. Vaughan, and A. Howard. The player/stage project: Tools for multi-robot and distributed sensor systems. In *In Proceedings of the 11th International Conference on Advanced Robotics*, pages 317–323, 2003.

[16] G. Grisetti, C. Stachniss, and W. Burgard. Improved techniques for grid mapping with rao-blackwellized particle filters. *IEEE Transactions on Robotics*, 23:2007, 2007.

[17] S. J. Guy, J. Chhugani, C. Kim, N. Satish, M. C. Lin, D. Manocha, and P. Dubey. Clearpath: Highly parallel collision avoidance for multi-agent simulation. In *ACM SIGGRAPH/Eurographics Symposium on Computer Animation (SCA)*, pages 177–187. ACM, 2009.

[18] D. Hennes, D. Claes, K. Tuyls, and W. Meeussen. Multi-robot collision avoidance with localization uncertainty. In *Proceedings of the International Conference on Autonomous Agents and Multiagent Systems (AAMAS)*, Valencia, Spain, June 2012.

[19] B. Kluge, D. Bank, E. Prassler, and M. Strobel. Coordinating the motion of a human and a robot in a crowded, natural environment. In *Advances in Human-Robot Interaction*, volume 14 of *Springer Tracts in Advanced Robotics*, pages 231–234. Springer Berlin / Heidelberg, 2005.

[20] B. Kluge and E. Prassler. Reflective navigation: Individual behaviors and group behaviors. In *Proceedings of the IEEE International Conference on Robots and Automation (ICRA)*, pages 4172–4177, 2004.

[21] Y. Koren and J. Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Proceedings of the IEEE International Conference on Robotics and Automation*, pages 1398 –1404 vol.2, apr 1991.

[22] M. Kraus and K. Bürger. Interpolating and downsampling rgba volume data. In *Proceedings of Vision, Modeling, and Visualization*, 2008.

[23] D. Mitzel, G. Floros, P. Sudowe, B. van der Zander, and B. Leibe. Real time vision based multi-person tracking for mobile robotics and intelligent vehicles. In *Proceedings of the International Conference on Intelligent Robotics and Applications (ICIRA)*, pages 105–115, 2011.

[24] M. Quigley et al. ROS: An open-source Robot Operating System. In *Proceedings of the Open-Source Software workshop (ICRA)*, 2009.

[25] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. Independent navigation of multiple mobile robots with hybrid reciprocal velocity obstacles. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5917–5922, 2009.

[26] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. Smooth and collision-free navigation for multiple robots under differential-drive constraints. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2010.

[27] J. Snape, J. van den Berg, S. J. Guy, and D. Manocha. The hybrid reciprocal velocity obstacle. *IEEE Transactions on Robotics*, 27(4):696–706, 2011.

[28] S. Thrun, W. Burgard, and D. Fox. *Probabilistic Robotics (Intelligent Robotics and Autonomous Agents series)*. The MIT Press, 2005.

[29] J. van den Berg, S. Guy, M. Lin, and D. Manocha. Reciprocal n-body collision avoidance. In *Robotics Research*, volume 70, pages 3–19, 2011.

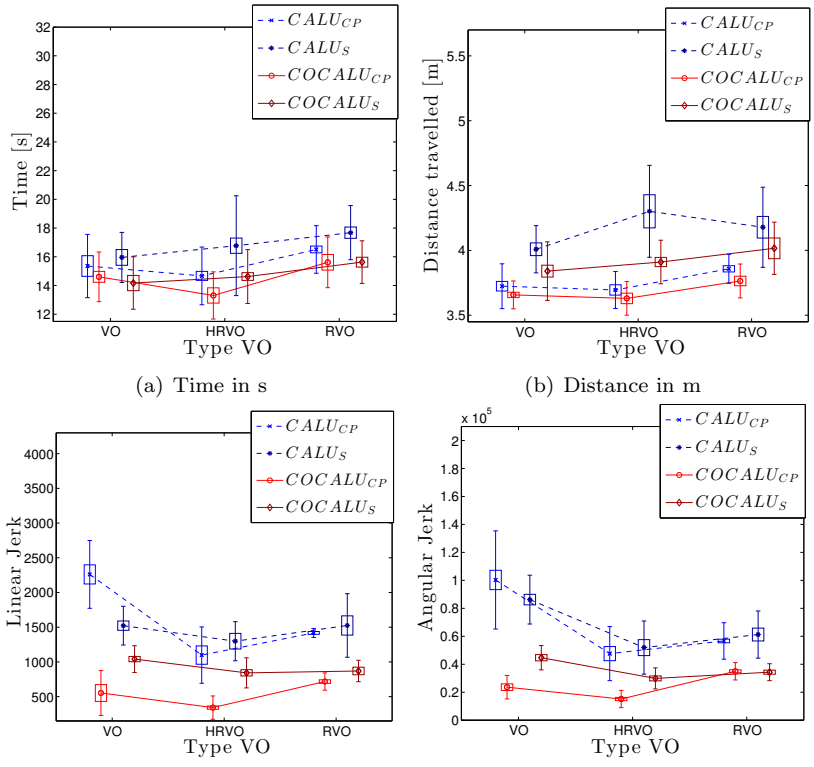
[30] J. van den Berg, M. Lin, and D. Manocha. Reciprocal velocity obstacles for real-time multi-agent navigation. In *Proceedings of the IEEE International Conference on Robots and Automation (ICRA)*, pages 1928 –1935, 2008.

[31] J. van den Berg, J. Snape, S. Guy, and D. Manocha. Reciprocal collision avoidance with acceleration-velocity obstacles. In *Proceedings of the IEEE International Conference on Robots and Automation (ICRA)*, 2011.

[32] R. Vaughan. Massively multi-robot simulation in stage. *Swarm Intelligence*, 2(2):189–208, 2008.

Appendix A

Results for VO selection with ground truth



(a) Time in s

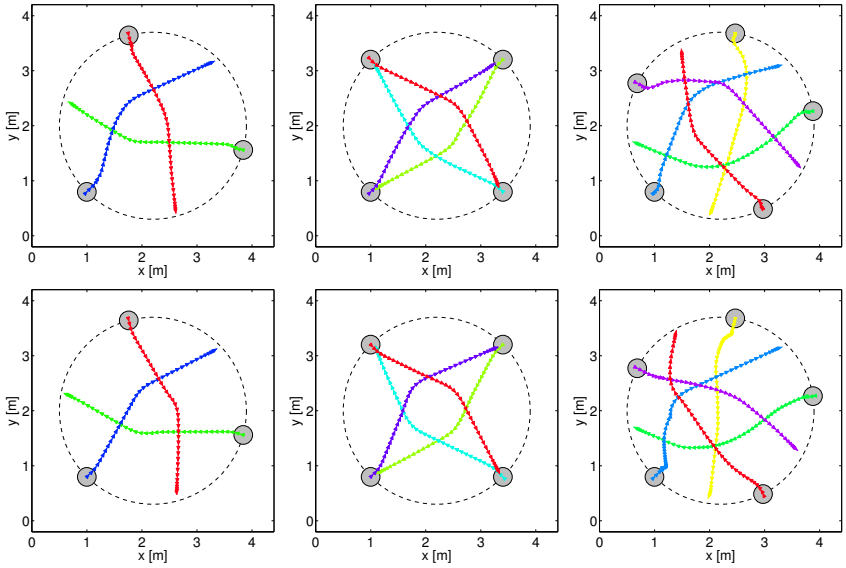
(b) Distance in m

(c) Linear Jerk

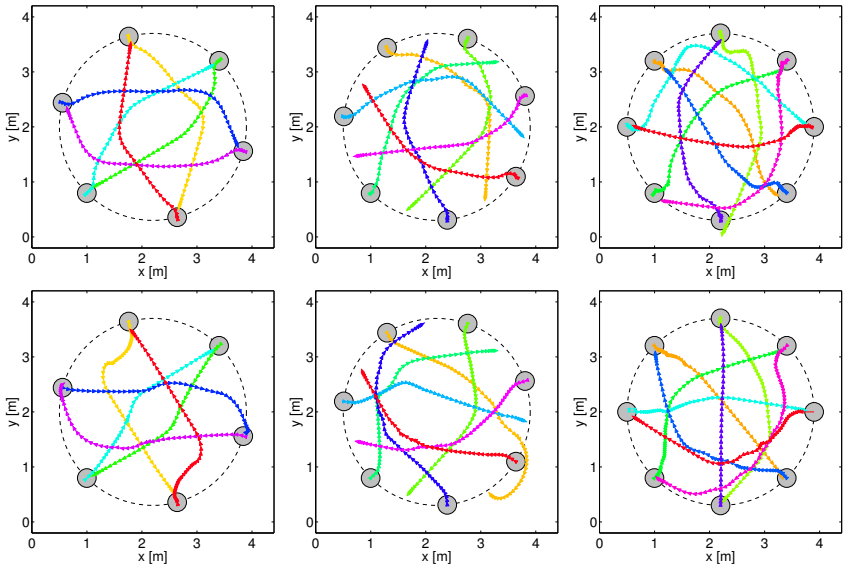
(d) Angular Jerk

Figure A.1: Comparing the different VO types for CALU and COCALU with ground truth. The plots are slightly offset to enhance comparability. The boxes define the 90% confidence interval for the mean and the bars show the standard deviation. (a) Average time needed to complete the run. (b) Distance travelled. (c) Linear jerk. (d) Angular jerk.

A.1 Simulation runs with round robots



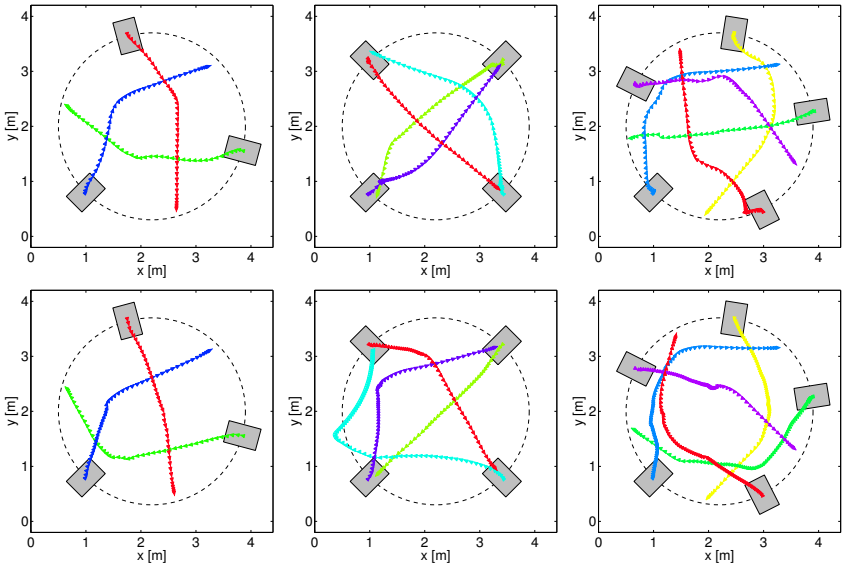
(a) $COCALU_{CP}$ (on top) and $CALU_{ORCA}$ below with $N = (3, 4, 5)$



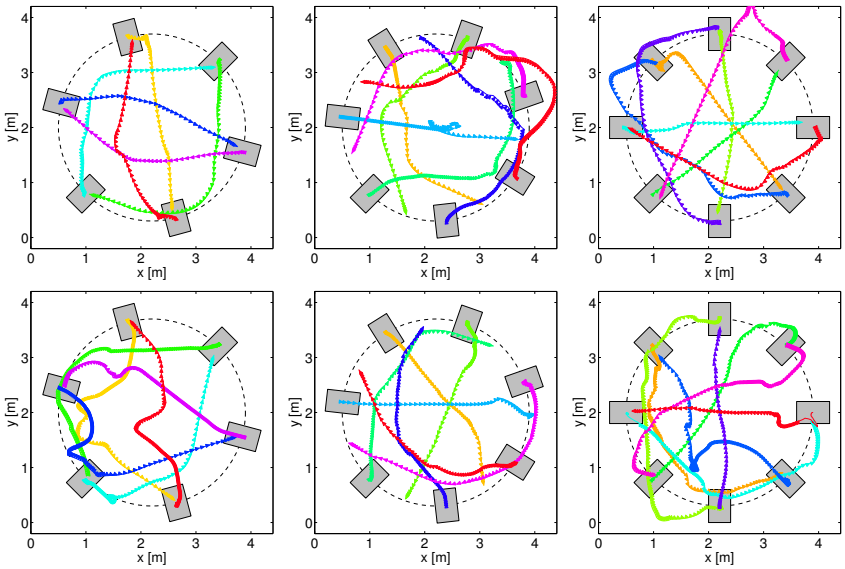
(b) $COCALU_{CP}$ (on top) and $CALU_{ORCA}$ below with $N = (6, 7, 8)$

Figure A.2: Sample trajectories with different numbers of circular differential drive robots using $COCALU_{CP}$ and $CALU_{ORCA}$ and AMCL.

A.2 Simulation runs with rectangular robots



(a) $COCALU_{CP}$ (on top) and $CALU_{ORCA}$ below with $N = (3, 4, 5)$



(b) $COCALU_{CP}$ (on top) and $CALU_{ORCA}$ below with $N = (6, 7, 8)$

Figure A.3: Sample trajectories with different numbers of rectangular holonomic robots using $COCALU_{CP}$ and $CALU_{ORCA}$ and AMCL.

Appendix B

Real world results

B.1 Round turtlebots

	Time [s]	Distance [m]	Linear Jerk	Angular Jerk
$COCALU_{ORCA}$	13.45 ± 1.82	3.03 ± 0.11	759.11 ± 165.23	8835.84 ± 2944.38
$CALU_{ORCA}$	13.36 ± 3.08	3.07 ± 0.11	761.87 ± 253.36	7828.40 ± 3603.39
$COCALU_{CP}$	14.16 ± 2.06	3.13 ± 0.09	761.56 ± 197.18	10271.57 ± 4404.64
$CALU_{CP}$	13.73 ± 1.34	3.07 ± 0.09	827.03 ± 190.86	10381.79 ± 2653.95
$COCALU_{S}$	13.27 ± 0.68	3.04 ± 0.04	810.46 ± 118.00	9594.88 ± 1879.60
$CALU_{S}$	12.86 ± 0.35	3.08 ± 0.02	882.17 ± 143.23	8740.49 ± 2200.62

Table B.1: Results for 10 runs in real life with round shaped turtlebots.

B.2 Quadratic shaped turtlebots

	Time [s]	Distance [m]	Linear Jerk	Angular Jerk
$COCALU_{ORCA}$	14.33 ± 1.21	2.71 ± 1.02	554.64 ± 234.19	7098.96 ± 3543.96
$CALU_{ORCA}$	16.33 ± 1.10	3.15 ± 0.13	657.17 ± 203.09	8953.85 ± 3516.70
$COCALU_{CP}$	14.94 ± 1.46	2.95 ± 0.97	616.17 ± 256.54	13467.98 ± 4909.67
$CALU_{CP}$	15.95 ± 1.50	3.14 ± 0.07	750.91 ± 130.04	9708.18 ± 5038.84
$COCALU_{S}$	14.08 ± 1.52	2.84 ± 0.93	869.05 ± 310.77	17660.79 ± 8299.73
$CALU_{S}$	14.18 ± 1.92	2.88 ± 0.94	768.18 ± 300.07	14187.81 ± 4707.93

Table B.2: Results for 10 runs in real life with quadratic shaped turtlebots.