University of Stuttgart
Germany

THE UNIVERSITY OF WESTERN AUSTRALIA

Master thesis

Implementation of Formation Control with Optimal Path Planning and Collision Avoidance

by
Marius Fink

in coorporation with
Robotics & Automation Lab of University of Western Australia, Australia

A thesis presented to the faculty in partial fulfillment of the requirements for the degree Master of Science in
Engineering Cybernetics

Examiner
Prof. Dr.-Ing. Dr. h.c. Oliver Sawodny

Supervisors
Prof. Dr. rer. nat. habil. Thomas Bräunl
M.Sc. Hannes Wind

Examination Date
2019-10-23

Institute for System Dynamics, University of Stuttgart, Waldburgstr. 17/19, 70563 Stuttgart

Abstract

This thesis investigates the implementation of formation control with a leader-follower approach for differentially driven robots. In this way, the positions of multiple robots are controlled such that they attain a specified formation. Furthermore, an optimal path planning algorithm is discussed and implemented. In combination with the leader-follower controller, the generated path is used as a virtual leader to navigate multiple robots along the desired trajectory. In the planning algorithm, the system dynamics of the robots and the avoidance of known obstacles is considered. Additionally, the formation controller is extended to incorporate on-board position sensitive devices in order to detect and avoid any obstacles close to the robot. All of these control schemes are first evaluated in simulation while measurement noise and subsequently in real experiments. The resulting combination of control schemes is able to navigate the robots to a goal position in an arbitrary world setup without any collisions.

Contents

Chapter 1

Motivation

In recent years, the interest in self driving cars has grown rapidly. Without the interaction of a user, a car is able to drive around and decide autonomously how to behave in different situations using artificial intelligence. This idea also motivates the development of a controller for solving similar tasks. Although in this case the goals of artifical intelligence and control theory are comparable, they vary greatly in their implementation. Artificial intelligence is based on an enormous amount of training data, while a controller uses a physical model to derive its control inputs.
Since the 1960s [1], solving a control problem for mobile robots earned major attention, due to their theoretical challenges and diverse applications, like explorations [2], rescue missions [3] or entertainment such as robot soccer games [4]. If a coordinated group of robots is used, many of these tasks could be executed faster, cheaper and more reliably [5].
Controlling the position of multiple robots while maintaining a desired spatial pattern is called formation control. These control problems are aiming to find a controller, which calculates automatically and continuously a control input, like a desired velocity or steering angle of a mobile robot. A survey regarding this research area is conducted in [6]. Formation control can be divided into three main approaches, the behavior-based approach [4], virtual structure [7] and leader-follower solution [8]. The behavior-based approach focuses on coordinating the behaviors of individual robots, while an approach with virtual structures considers the formation of multiple robots as a single object. A leader-follower approach defines at least one robot as the leader and the remaining robots are assigned to follow the leader. Due to its simplicity and scalability, the leader-follower approach is widely adopted and has been studied extensively [9]. This approach is investigated throughout the course of this thesis.
It is desirable to identify a continuous path for the robots in order to preplan their navigation in a smooth fashion. This is addressed with a path planning algorithm, which generates a path from a start to a goal, while additional constraints can be adressed. An overview about different algorithms is given in [10], where the various solutions are divided into the categories roadmap techniques [11], cell decomposition algorithms [12] and artificial potential methods [13]. The roadmap technique maps the available space into a system of one-dimensional curves. This approach is specifically used for high dimensional problems. A cell decomposition method divides the total free space into several regions and the algorithm is looking for a sequence of cells to reach the target. An artificial potential approach extends the control scheme for a function which repels the robot of any unwanted location. Avoiding obstacles is one core element for all of these techniques.
The goal of this thesis is the implementation of a formation controller with a path planning algorithm, that is capable of navigating multiple robots in a world with several obstacles. To reduce the difference of the simulation to the experiments, uncertainties regarding the knowledge of the world and sensor measurements is considered during the simulation.

Chapter 2

Background

This chapter introduces the basics needed for the later analysis in chapter 3. At first, a differentially driven robot is defined. Next, the research project of this thesis and the simulation software, used to evaluate the functionality of the control schemes, are presented.
Lastly, a general concept for a path planning algorithm is introduced, which finds an optimal path without any collision with an obstacle.

2.1 Differentially Driven Robot

One of the most popular architecture of mobile robots is a so called differentially driven robot [14]. These are car-like robots, which have been extensively studied due to their simple functionality and broad application area.

2.1.1 System Dynamics

A generic sketch of a differentially driven vehicle is depicted in fig. 2.1. In general, motors are connected to the left and right wheel of the robot. Additionally, the robot has a third wheel, which is called a castor c, and prevents the robot from falling over. It is placed either in the front or the rear [15].
The state of a differentially driven robot $q(t) = (x(t), y(t), \theta(t))^T$ consists of the position variables $x(t)$, $y(t)$ and the steering angle $\theta(t)$. The state vector, which does not include $\theta(t)$, is denoted with $\tilde{q}(t) = (x(t), y(t))^T$. For better readability, the time argument of the variables is omitted in obvious cases in this work.
The dynamic equations of a differentially driven robot are

$$\begin{aligned} \dot{x} &= r\left(\frac{\omega_L + \omega_R}{2}\right) cos(\theta) \\ \dot{y} &= r\left(\frac{\omega_L + \omega_R}{2}\right) sin(\theta) \\ \dot{\theta} &= r\left(\frac{\omega_L - \omega_R}{b}\right), \end{aligned} \tag{2.1}$$

where $r > 0$ denotes the wheel radius and $b > 0$ the distance between the wheels. ω_L and ω_R are the inputs of the system and describe the angular velocities of the left and right

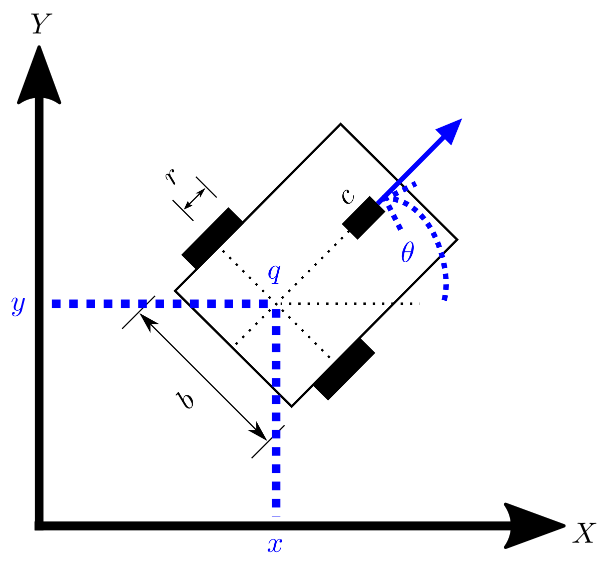


Figure 2.1: Visualization of a differentially driven vehicle, adapted from [14].

wheel respectively [16]. If the transformation

$$u = \begin{pmatrix} v \\ \omega \end{pmatrix} = T \begin{pmatrix} \omega_L \\ \omega_R \end{pmatrix} = \begin{bmatrix} r/2 & r/2 \\ r/b & -r/b \end{bmatrix} \begin{pmatrix} \omega_L \\ \omega_R \end{pmatrix}, \tag{2.2}$$

is applied, the input u changes to a more intuitive representation: the linear velocity v and the angular velocity ω of the vehicle. Now, the differentially driven model simplifies to

$$\begin{aligned} \dot{x} &= vcos(\theta) \\ \dot{y} &= vsin(\theta) \\ \dot{\theta} &= \omega. \end{aligned} \tag{2.3}$$

These equations also represent the dynamics of a unicycle, if the balancing concerns are neglected [14]. Since the matrix T is always invertible, it suffices to analyze eq. (2.3) for deriving a control law [16].
The exact discrete time dynamics of this system can be attained by using direct integration, when the control inputs are constant on each interval $[kt_s, (k+1)t_s), k \in \mathcal{N}_0$ [17]. The equation is given by

$$q(k+1) = f_{t_s}(q(k), u(k)) = \begin{pmatrix} x(k) \\ y(k) \\ \theta(k) \end{pmatrix} + \begin{pmatrix} \frac{v(k)}{\omega(k)}(\ \ sin(\theta(k) + t_s\omega(k)) - sin(\theta(k))) \\ \frac{v(k)}{\omega(k)}(-cos(\theta(k) + t_s\omega(k)) + cos(\theta(k))) \\ t_s\omega_k \end{pmatrix} \tag{2.4}$$

with sampling time t_s and $\omega(k) \neq 0$.
When moving in a straight line, the right hand side of eq. (2.4) becomes

$$q(k) + \lim_{\omega \to 0} \begin{pmatrix} \frac{v(k)}{\omega(k)}(\ \ sin(\theta(k) + t_s\omega(k)) - sin(\theta(k))) \\ \frac{v(k)}{\omega(k)}(-cos(\theta(k) + t_s\omega(k)) + cos(\theta(k))) \\ t_s\omega_k \end{pmatrix} = q(k) + t_s v(k) \begin{pmatrix} cos(\theta(k) \\ sin(\theta(k)) \\ 0 \end{pmatrix} \tag{2.5}$$

and therefore the case $\omega(k) = 0$ has to be considered separately.

In contrast to an automobile, a differentially driven robot can turn on the spot, if the linear velocity $v = 0$ and angular velocity $\omega \neq 0$ holds. However, these robots are still not able to drive sideways. This restriction is expressed by the nonholonomic no-slip constraint [18]

$$A(q)\dot{q} = 0 \text{ with } A(q) = [-sin(\theta),\ cos(\theta),\ 0]. \tag{2.6}$$

During a path planning algorithm this restriction has to be considered to result in a smooth and consistent trajectory.

2.1.2 EyeBot Project and EyeSim

For the last 20 years, the EyeBot Project [19] focused on the research of embedded controllers for robotics applications. This includes the development of mobile robots and simulating their behavior with the multi-robot, multi-tasking simulation software EyeSim. The project is researching various different types of robots, like omni-directional robots, underwater vehicles or flying robots. The current focus lies on the previously introduced differentially driven robots. The EyeBot robots were originally designed to compete in Robosoccer [20], but now the research area has been extended e.g. to image processing and deep learning [21].
The differentially driven robots are based on the operating system RoBIOS (robot basic input output system) of EyeBot and they are programmable in C or Python. Amongst other things, RoBIOS provides access to various driving functions, realistic sensor readings and communication between robots via a radio function. For driving, the user can choose between high level functions, e.g. driving straight or following a curve segment for a set distance and angle, and low level functions, e.g. setting the linear and angular velocity or actuate the motors directly. An EyeBot robot is mounted with position sensitive devices (PSD) for measuring the distance towards an object. The software provides the possibility to consider noise for the position readings and the PSD sensors. Thus, a better representation of the real sensors can be achieved. During the simulation, any collision of a robot is detected. One major advantage of the project is the combination of EyeSim with the EyeBot robots. Thus, it is possible to evaluate and test any algorithm first in simulation, and at a later stage, these algorithms can be implemented on the real robots without any significant changes.
The simulation of an EyeBot robot is depicted in fig. 4.1. The Eyesim software is used for the simulations presented in chapter 3 and the experiments in chapter 4 are conducted with the EyeBot robots.

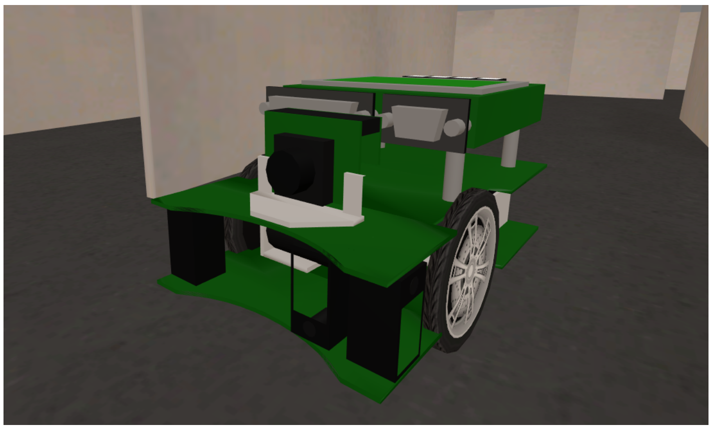


Figure 2.2: Simulation of a differentially driven robot with EyeSim.

2.2 Path Planning

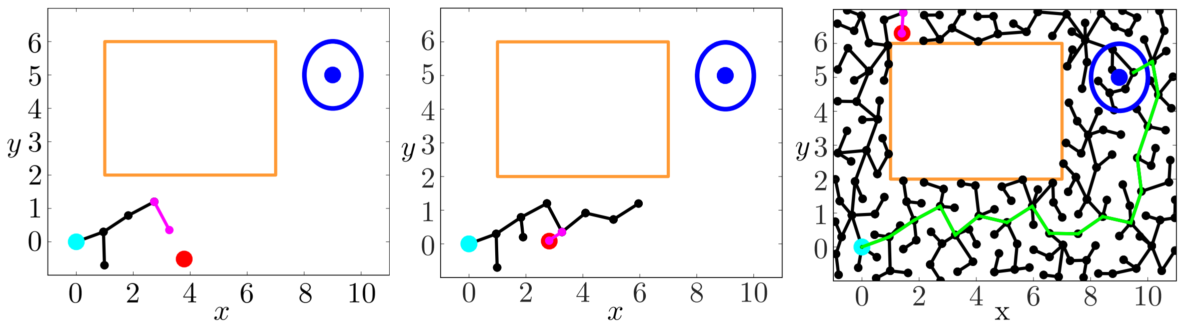
A path planning algorithm determines a sequence of configurations to navigate a robot from a start to a goal position, without interfering with any known obstacles. One famous algorithm is the Rapidly Exploring Random Tree, which will be introduced and analyzed in this section and later in section 3.2.1, it will be extended to satisfy the kinematics of a unicycle.

2.2.1 Rapidly Exploring Random Tree

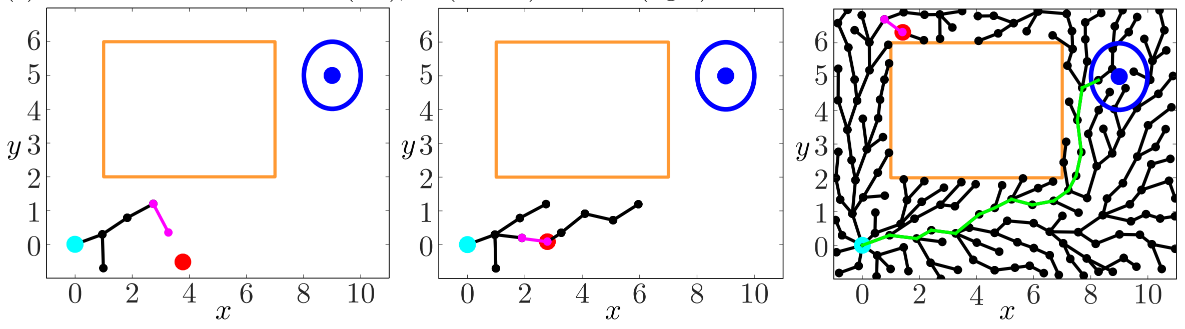
A Rapidly Exploring Random Tree (RRT) is first introduced by LaValle in [22] and can be used for a broad class of path planning problems. A randomized data structure $\mathcal{T}$ is created in order to find a path from a start position $\tilde{q}_{start}$ to a goal $\tilde{q}_{goal}$, while additional conditions, e.g. not being obstructed by obstacles, can be addressed. The RRT-Star algorithm proposed by Karaman in [23] extends this method to guarantee asymptotic optimality.
First, a random point $\tilde{q}_{rand}$ is selected within some bounded area. The closest node to $\tilde{q}_{rand}$ within $\mathcal{T}$ becomes $\tilde{q}_{parent}$ and a path is created, if the length of this path is not smaller than some value d_{min}. $\tilde{q}_{new}$ is chosen such that it either becomes $\tilde{q}_{rand}$ or, if the path is longer than d_{max}, it steers into the direction of $\tilde{q}_{rand}$ until the threshold is reached. Checking that $\tilde{q}_{rand}$, $\tilde{q}_{new}$ and the connecting path do not interfere with any obstacles $\tilde{q}_{obst}$ is required during these steps. Finally, the new node $\tilde{q}_{new}$ is added to the RRT $\mathcal{T}$. Since it is very unlikely that $\tilde{q}_{goal}$ is exactly reached, a goal region must be defined such that it is sufficient if $\tilde{q}_{new}$ lies within that region.
The RRT-Star algorithm extends the choice of $\tilde{q}_{parent}$ and rewires $\mathcal{T}$. After $\tilde{q}_{new}$ is chosen, $\tilde{q}_{parent}$ is re-selected as the best node in $\mathcal{T}$, which can reach $\tilde{q}_{new}$. In order to identify the cheapest neighbour, a state requires to have a cost function with respect to the total traveled distance, time or any other measure. Secondly, after each iteration it is checked whether any node within the neighbourhood gets a lower total cost with $\tilde{q}_{new}$. If so, the parent of this node is changed to $\tilde{q}_{new}$. A summary of the basic steps to generate a RRT, including the RRT-Star extensions, are stated in algorithm 1.

Algorithm 1 Pseudo Code for a Rapidly Exploring Random Tree $\mathcal{T}(x_{init}, d, \tilde{q}_{obst}, K_{max})$ [22] with star extensions [23].

```
1: initialize $\mathcal{T}(\tilde{q}_{init})$
2: for n = 1 to $N_{max}$ do
3:     $\tilde{q}_{rand} \leftarrow$ random point
4:     $\tilde{q}_{parent} \leftarrow$ closest neighbour $(\tilde{q}_{rand}, d, \tilde{q}_{obst}, \mathcal{T})$
5:     $u \leftarrow$ select input $(\tilde{q}_{rand}, \tilde{q}_{parent}, d)$
6:     $\tilde{q}_{new} \leftarrow$ new node $(\tilde{q}_{parent}, u, \tilde{q}_{obst})$
7:     $\tilde{q}_{parent} \leftarrow$ cheapest neighbour $(\tilde{q}_{new}, d, \tilde{q}_{obst}, \mathcal{T})$
8:     $\mathcal{T} \leftarrow$ add new node $(\tilde{q}_{new})$
9:     $\mathcal{T} \leftarrow$ add new path $(\tilde{q}_{new}, \tilde{q}_{parent}, u)$
10:    $\mathcal{T} \leftarrow$ rewire graph $(\tilde{q}_{new}, \tilde{q}_{neighbours}, u)$
11: end for
12: return $\mathcal{T}$
```



(a) Results of RRT with n = 5 (left), 10 (middle) and 200 (right) nodes.



(b) Results of RRT-Star with n = 5 (left), 10 (middle) and 200 (right) nodes.

Figure 2.3: Comparison of the optimal path of RRT and RRT-Star with $\tilde{q}_{start}$, $\tilde{q}_{goal}$, $\tilde{q}_{obst}$, $\tilde{q}_{rand}$, $\tilde{q}_{new}$, $\tilde{q}_{node}$ and optimal path.

A comparison of these two different variations is visualized in fig. 2.3. For the first $n = 5$ steps, the RRT in fig. 2.3a and the RRT-Star in fig. 2.3b are equivalent. A difference is visible at $n = 10$. Based on the proximity to $\tilde{q}_{new}$ and not the total traveled distance, the RRT is selecting a different $\tilde{q}_{parent}$ for the same $\tilde{q}_{new}$ than the RRT-Star algorithm. Additionally, rewiring of the graph causes to change the parent of $\tilde{q}_{new}$'s neighbour and therefore the overall tree structure becomes smoother. This smoothness can be particularly seen at $n = 200$, where two structurally different trees have been growing and the optimal path of RRT-Star is shorter than the path of RRT.
The tree structure of RRT-Star induces mathematical optimality, when $N_{max} \to \infty$ and the difference between each node $d_{min} \to 0$. However, the optimality properties cause increased complexity [23]. In the RRT-Star algorithm, every node within $\mathcal{T}$ has to be scanned multiple times and therefore the total computation time increases exponentially with the number of nodes. Both methods can be used for arbitrary system dynamics while satisfying nonholonomic constraints [23]. This possibility is further investigated in section 3.2.1.

2.2.2 Homogenous Transformation

In order to describe the position of mobile robots, the ability to change their coordinates with respect to different coordinate systems is indispensable. In general, the position of a global coordinate frame is fixed and it can be used to describe a setup for multiple robots. On the other hand, every robot holds its own local coordinate system, where the robot itself is always at the origin and the coordinate frame is used to distinguish the location of an object relative

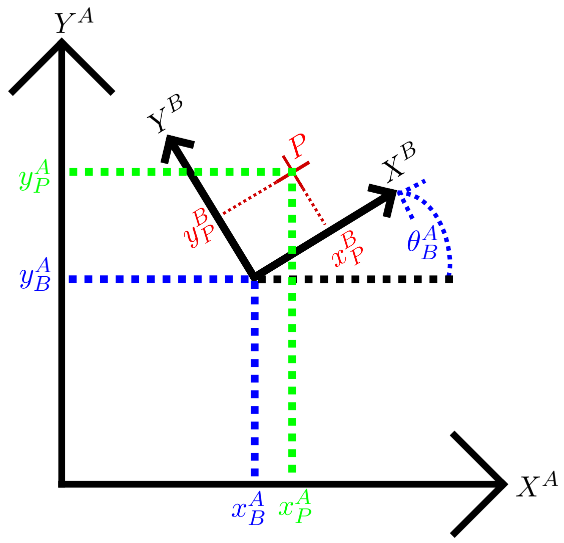


Figure 2.4: Relation between coordinate system A and B for coordination transformation.

to the corresponding robot. The relation between coordinate system A and B is visualized in fig. 2.4. The coordinates of point P are denoted as (x_P^B, y_P^B), where the superscript indicates the affiliation to system B. Using a rotational and translational coordinate transformation[1] [24] , P is transformed from system B into system A with

$$\begin{aligned} x_P^A &= cos(\theta_B^A)x_P^B + sin(\theta_B^A)y_P^B - x_B^A \\ y_P^A &= -sin(\theta_B^A)x_P^B + cos(\theta_B^A)y_P^B - x_B^A. \end{aligned} \tag{2.7}$$

x_B^A, y_B^A and θ_B^A are the pose of the coordinate frame B expressed with respect to A. To result in a matrix notation for better computability, the position coordinates can be extended with an additional column to

$$\begin{pmatrix} x_P^A \\ y_P^A \\ 1 \end{pmatrix} = \begin{pmatrix} cos(\theta_B^A) & sin(\theta_B^A) & -x_B^A \\ -sin(\theta_B^A) & cos(\theta_B^A) & -y_B^A \\ 0 & 0 & 1 \end{pmatrix} \begin{pmatrix} x_P^B \\ y_P^B \\ 1 \end{pmatrix}, \tag{2.8}$$

also known as homogenous coordinates.
With this approach, it is possible to transform the position and any information of a robot into different coordinate systems.

[1]In the literature, these equations usually state the inverse transformation. To be consistent with section 3.1 and due to the fact that the direction is solely a matter of definition, the inverse is neglected here.

Chapter 3

Simulation Results

The first step of controlling robots through a world with obstacles, is finding a suitable control scheme. This chapter will verify, that the leader-follower tracking controller by Loria et al. [25] can be used to fulfill this task. Additionally, an optimal path planning algorithm is used to create a path for a unicycle robot and a method to change the control scheme in order to maneuver around obstacles will be proposed. The control scheme will be the basis for all of these extensions.
Every section first introduces the theoretical basics and is concluded with simulation results, which verify their functionality. All of the simulations are conducted with EyeSim.
At the end of this chapter, every algorithm is combined to navigate two robots through an extended world with known and unknown obstacles.

3.1 Leader-Follower Tracking Control

As stated in the motivation, a leader-follower approach will be investigated throughout the course of this thesis. In [26], several leader-follower approaches are compared and experimentally evaluated. In [5], the leader-follower relation is defined with a cluster framework to control and monitor formation parameters relevant to the specific task. A large number of robots are capable of achieving a complex formation, if the control scheme of [27] is followed. The controller is based on a consensus algorithm and one advantage of this approach is the automatic dividing of the robots into leader and follower. Especially during the earlier proposals, it is often differentiated between driving along a curve or a straight line due to limitations in the stability proof. Covering both cases is crucial for following an arbitrary trajectory and therefore must be considered when choosing a control scheme. Many of the proposals in the literature focus on finding full state feedback controllers with global knowledge.
The controller used throughout this thesis, is proposed in [25] and originally based on [28], where results from nonlinear and adaptive control are combined. Its main purpose is to control the position of a follower, such that a leader is followed with a desired displacement in $x-$ or $y-$coordinates. In contrast to the aforementioned implementations, the leader follower tracking controller defines the desired configuration of the leader in global coordinates, not in local. Moreover, this controller can be used to follow a trajectory or as a point-to-point position controller. During the simulation, inaccurate sensor readings are used.

3.1.1 Control Algorithm

As introduced by the unicycle model in eq. (2.3), the dynamic equations of N mobile robots are given by

$$\begin{aligned} \dot{x}_i &= v_i cos(\theta_i) \\ \dot{y}_i &= v_i sin(\theta_i) \\ \dot{\theta}_i &= \omega_i \end{aligned} \tag{3.1}$$

with $i \in [0, .., N]$. The goal of every robot i is to follow its leader $i-1$ with a displacement $d_{d,i} = (d_{d,x,i}, d_{d,y,i})^T$ in global coordinates and where $i = 0$ implies a virtual leader. Therefore, the desired configuration for the follower is

$$\begin{aligned} x_{d,i} &= x_{i-1} - d_{d,x,i} \\ y_{d,i} &= y_{i-1} - d_{d,y,i} \\ \theta_{d,i} &= \theta_{i-1} \end{aligned} \tag{3.2}$$

and the controller's objective is to drive the cartesian error coordinates

$$\begin{aligned} p_{x,i} &= x_{d,i} - x_i \\ p_{y,i} &= y_{d,i} - y_i \\ p_{\theta,i} &= \theta_{d,i} - \theta_i \end{aligned} \tag{3.3}$$

to zero. Transforming these equations into local error coordinates of the follower results in

$$\begin{aligned} e_{x,i} &= cos(\theta_i) p_{x,i} + sin(\theta_i) p_{y,i} \\ e_{y,i} &= -sin(\theta_i) p_{x,i} + cos(\theta_i) p_{y,i} \\ e_{\theta,i} &= p_{\theta,i}. \end{aligned} \tag{3.4}$$

In [29] it is shown that the inputs

$$\begin{aligned} v_i &= v_{i-1} + c_a e_{x,i} \qquad && c_a > 0 \\ \omega_i &= h(t, e_{y,i}) + c_b e_{\theta,i} \qquad && c_b > 0 \end{aligned} \tag{3.5}$$

lead to an uniformly globally asymptotically stable origin, if h is bounded, locally of linear order in $e_{y,i}$ and continuously differentiable. c_a and c_b are control gains and must be positive. Here, h replaces the original condition $\omega_{i-1} \neq 0$ and implies that $\dot{h}$ has to be persistently exciting.
According to Loria et al. [25], the controller can be chosen to

$$\begin{aligned} v_i &= v_{i-1} + c_a e_{x,i} \\ \omega_i &= \omega_{i-1} + c_c \phi(t) tanh(e_{y,i}) + c_b e_{\theta,i}, \qquad && c_c > 0, \end{aligned} \tag{3.6}$$

and for this case, the stability proof also covers straight lines. $\phi(t)$ is required to guarantee that the signal $h(t, e_{y,i})$ is uniformly δ-persistently exciting. This can be provided with $\phi(t)$ being a square-pulse-train signal. Even if $\omega_{i-1} = 0$ holds, the error dynamics are stabilized in y-direction, as for $e_{y,i} \neq 0$ the term $\phi(t) tanh\,(e_{y,i})$ induces enough excitation [29].
The stability proof is based on Lyapunov's Theory and the Small-Gain Theorem for input to state stable systems [29], where the unicycle model is used to derive the closed loop error

kinematics. During the stability analysis of [29], it is proven that the control law of eq. (3.6) leads to an asymptotically stable origin for the kinematics of the error coordinates e. This means, that $e(t) \to 0$ for $t \to \infty$ and as the error converges to zero, the measured position q_i converges to the desired position $q_{d,i}$ and stays there eventually. It is important to notice that only boundedness of v_{i-1} is required. Therefore, this controller can also be used as a point-to-point position controller, if the (virtual) leader is set to the desired coordinates and $v_{i-1} = \omega_{i-1} = 0$ holds. For an in-depth analysis of the theoretical results, refer to the original sources [28], [25] and [29].

3.1.2 Simulation

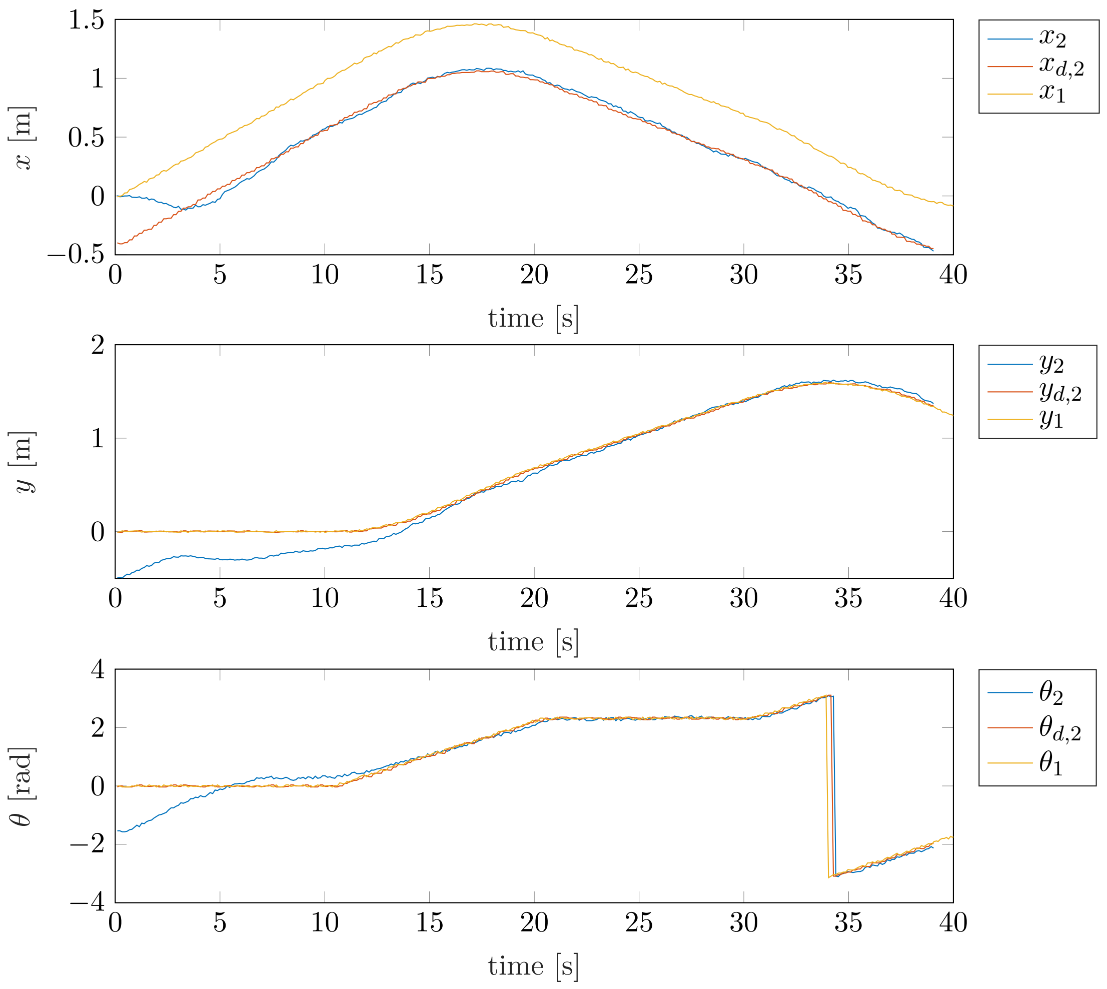
This section investigates the leader-follower tracking controller. First, a scenario, where the controller is used to follow a robot, is presented. Afterwards the effect of the measurement noise is analyzed.
The leading robot 1 is driving along a trajectory, which is divided in four segments, while robot 2 is assigned to follow its leader with a chosen offset. The trajectory segments last for 10 seconds and they are changing from driving straight to driving a curve. In order to use the control law of eq. (3.6), continuous communication between the leader and the follower is required to compute the control inputs $u_2 = (v_2, \omega_2)$. The follower must receive a broadcast from the leader, containing the whole state information q_1 and the input signals u_1. Additionally, the initial positions for both robots have to be known. They are arbitrarily chosen to

$$q_1(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, q_2(0) = \begin{pmatrix} 0 \\ -0.5 \\ -\frac{\pi}{2} \end{pmatrix} \tag{3.7}$$

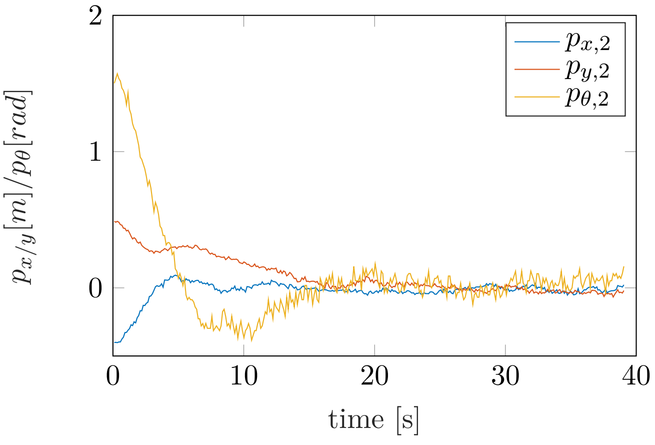
and the control parameters are

$$d_{d,x,2} = -0.5, d_{d,y,2} = 0 \quad \text{and} \quad c_a = 0.5, c_b = 0.5, c_c = 1, \tag{3.8}$$

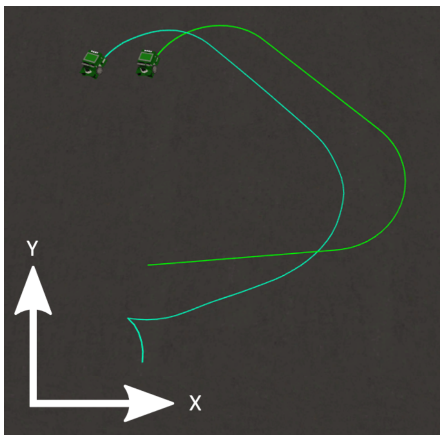
implying that the follower tracks its leader with a displacement in $x-$direction and no offset in $y-$coordinates. The square-pulse-train signal $\phi(t)$ has a period of four seconds, a pulse width of 3 seconds and an amplitude of 1.
The result of the controller is visualized in fig. 3.1. Figure 3.1a shows the state-space coordinates $q = (x, q, \theta)$ of both robots and the desired goal configuration for the follower $q_{d,2}$. The follower converges through a transition phase to $q_{d,2}$ and then follows its leader with the desired displacement in $x-$direction, while the desired $y-$coordinate are identical to the leader's. After 15 seconds the desired configuration is reached and the leader is followed with the specified offset, even if the leader's trajectory segment changes. It is mentionable, that the $y-$coordinates of the follower are stabilized for both $\omega_1 = 0$ and $\omega_1 \neq 0$, as incorporated during the control design. Analyzing the cartesian error coordinates p_2 in fig. 3.1b leads to the conclusion that the controller offers rapid response and good performance, as p_2 are converging to zero and stay there. Thus, the controller achieves its objective.
When the path of the robots in fig. 3.1c is analyzed, the impression that the follower is not satisfying the nonholonomic constraints at the beginning of the simulation might arise. However, if the initial position and especially $\theta_2(0) = -\frac{\pi}{2}$ are considered, it is noticable that the follower is facing in the opposite direction of $q_{d,2}$ for the first few seconds. Therefore, initially,



(a) State information q_2, desired coordinates $q_{d,2}$ of the follower and q_1 of the leader.



(b) Error coordinates p of the follower.



(c) Path of the leader and follower.

Figure 3.1: Simulation results of the leader-follower tracking controller with EyeSim.

the robot drives backwards while turning to reach $q_{d,2}$, as indicated by the partial circle at the beginning of the path. After the leader has advanced, the direction of the follower changes, as the robot moves forwards to reach $q_{d,2}$.
Using the control scheme as a point-to-point position controller is skipped in this section, as this possibility is used during the investigation of the collision avoidance extension in section 3.3.1.
As outlined in chapter 1, the controller must accomplish its objective while measurement noise is considered. In the simulation just mentioned, the position readings are disturbed by the inbuilt noise of EyeSim. The effect of the noise can be seen particularly in fig. 3.1b. As stated above, the controller achieves its objective in the perturbed case. Therefore, the impact of measurement noise with respect to the control inputs will be investigated further. An arbitrary state of the preceding scenario results in the nominal case of v_0 and ω_0. This state is copied, by holding all variables constant except of the steering angle θ. The disturbance of the leader's steering angle θ^{lead} has a different sign than θ^{follow} of the follower. Hence, the inputs $v_{-0.05}/\omega_{-0.05}$ for the case $\theta_{-0.05}$ use $\theta^{lead}_{-0.05} = \theta^{lead}_0 - 0.05$ and $\theta^{follow}_{-0.05} = \theta^{follow}_0 + 0.05$ and vice versa for $\theta_{0.05}$. As seen in fig. 3.2, the effect of varying θ towards v is neglectable. On the other hand, the variation of θ lies within the expected limit of $-0.55\frac{rad}{s}/-30\frac{\circ}{s}$ to $-0.45\frac{rad}{s}/-25\frac{\circ}{s}$, since θ^{lead} and θ^{follow} are varied for similar values. This leads to the conclusion that the controller is not amplifying the disturbance.

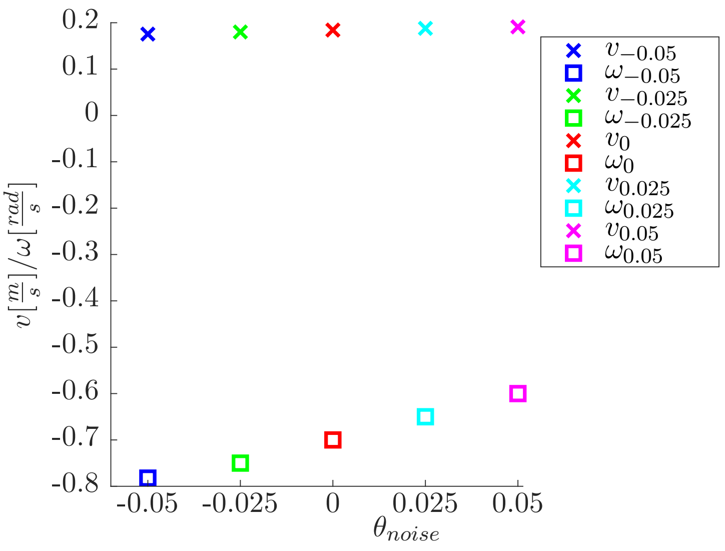


Figure 3.2: Comparison of the inputs for perturbed and unperturbed θ for the leader-follower tracking controller.

3.2 Optimal Path Planning

Optimal path planning algorithms are searching for a feasible and optimal path from a start to a goal, without hitting any known obstacles. For an EyeBot robot to follow this path, the unicycle dynamics and nonholonomic constraints of 2.1.1 must be satisfied and therefore considered during the algorithm. The RRT-Star algorithm, introduced in section 2.2.1, can be extended to follow system dynamics and handle additional constraints and high degrees of freedom [22]. This option is analyzed and the resulting path will be followed by a robot with the leader-follower tracking controller.

3.2.1 Dynamic RRT-Star

The dynamic RRT-Star algorithm is looking for a path, which satisfies the system dynamics of a unicycle (eq. (2.3)) and the nonholonomic no-slip constraint (eq. (2.6)). However, addressing these restrictions during the algorithm results in a complex and computationally intensive problem.
First, a definition for optimality in this context is required. Driving from one node to another takes a specific amount of time to accomplish. Therefore, the total time of the subsequent steps while traveling along the path, is minimized with the dynamic RRT-Star algorithm.
[30] suggests to define motion primitives to reduce the aforementioned computational challenges. Figure 3.3 shows three possible motion primitives, namely driving left, straight and right. The basic structure of the dynamic RRT-Star algorithm is unchanged and still follows algorithm 1, but now the algorithm is looking for the best sequence of these primitives to reach the goal.

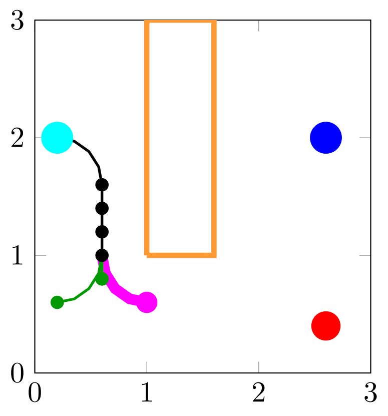


Figure 3.3: Possible motion primitives for the dynamic RRT-Star algorithm with q_{start}, q_{goal}, q_{Obst}, q_{rand}, q_{node} and q_{new}.

The primitives are created by choosing appropriate inputs for the linear and angular velocity and the duration for every motion. To follow the unicycle dynamics, every node has to be extended for a steering angle θ. Consequently, the same $x-/y-$position can be visited multiple times for different θ and additional checking during the neighbour selection and the rewiring is required to guarantee, that the overall path achieves a continuous θ.

For every new random point q_{rand}, the dynamic RRT-Star algorithm checks which primitive within $\mathcal{T}$ gets closest to the new point. When a new q_{new} is chosen, the algorithm has to find a position and orientation that is not yet present within $\mathcal{T}$. While checking for a cheaper possibility, it must be ensured that the orientation is not changed or otherwise a whole path can be corrupted. Special attention must be paid to the iterative calling of the rewiring function, as it is required to check whether any node with an updated cheaper cost can reach another new node with a cheaper cost now. The previous checking of the path not interfering with any obstacle is still required, but now it is extended to a minimal safety distance d_{Safety} between every node and the obstacle. Thus, the node cannot get arbitrarily close to the obstacle and it can be guaranteed that a region around the robot is also not colliding with an obstacle.
Discretizing the state space limits the number of possibilities and reduces the computational effort. Also, using the discretized version of the unicycle dynamics (eq. (2.4)) avoids the continuous integration of the system dynamics, which can get challenging, if the algorithm is implemented in C. Additionally, the necessity of introducing a goal region becomes obsolete. To stay consistent, the created random points match the discretized state space.
The generated path can be used as a virtual leader in combination with the leader follower tracking controller of section 3.1 to navigate a robot to a desired location in a time optimal way, while inherently avoiding any known obstacles.

3.2.2 Simulation

The option to combine the dynamic RRT-Star algorithm with the leader-follower tracking controller will be examined below. The motion primitives are chosen to be driving left, straight and right. The individual paths are created by applying

$$\begin{aligned} v_{left} &= \frac{\pi}{40} & v_{straight} &= \frac{2}{15} & v_{right} &= \frac{\pi}{40} \\ \omega_{left} &= \frac{\pi}{16} & \omega_{straight} &= 0 & \omega_{right} &= -\frac{\pi}{16} \\ t_{left} &= 8 & t_{straight} &= 1.5 & t_{right} &= 8 \end{aligned} \tag{3.9}$$

to the unicycle kinematics. v and ω denote the input to the robot and t is the duration of a primitive. These values are chosen such that the endpoints

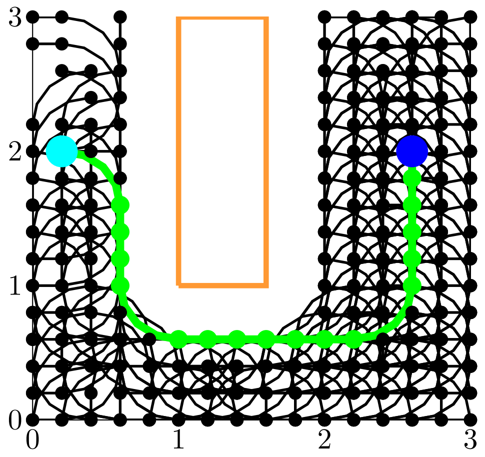
$$\begin{aligned} q_{left,end} &= \begin{pmatrix} x^{loc}_{start} \\ y^{loc}_{start} \\ \theta^{loc}_{start} \end{pmatrix} + \begin{pmatrix} 0.4 \\ 0.4 \\ \frac{\pi}{2} \end{pmatrix} \\ q_{straight,end} &= \begin{pmatrix} x^{loc}_{start} \\ y^{loc}_{start} \\ \theta^{loc}_{start} \end{pmatrix} + \begin{pmatrix} 0.2 \\ 0 \\ 0 \end{pmatrix} \\ q_{right,end} &= \begin{pmatrix} x^{loc}_{start} \\ y^{loc}_{start} \\ \theta^{loc}_{start} \end{pmatrix} + \begin{pmatrix} 0.4 \\ -0.4 \\ -\frac{\pi}{2} \end{pmatrix} \end{aligned} \tag{3.10}$$

fit to the discretization of the state space.

These coordinates are expressed in local coordinates. $x^{loc}_{start}, y^{loc}_{start}$ and θ^{loc}_{start} are the starting point, whereas q_{end} describes the new pose of the robot after a primitive has finished.
Although the unicycle allows to turn the robot on the same spot, this possibility is intentionally omitted to result in a smooth trajectory without abrupt transitions or even stopping the robot. The time value of driving straight is chosen to be significantly faster than turning left or right. To point this reasoning out, two fictive nodes are connected. Those nodes result in a triangle with the catheti being the $x-$ and $y-$axis, while the connecting line is the shortest distance and therefore the hypotenuse of the triangle. It is favorable, if the time values are chosen such that following the catheti is faster than turning consecutively along the hypotenuse. If the total time would be proportional to the traveled distance, the resulting path would get curvier. Obviously, driving straight for a longer distance is preferably over the latter case and with the chosen values a curve is only performed, if a change in θ is required
Revisiting the optimality property of the RRT-Star algorithm from section 2.2.1, the path converges to the true optimum for $N_{max} \to \infty$ and $d_{min} \to 0$. Therefore the distance of driving straight is shorter than the other primitives.
The algorithm was developed with MATLAB 2018b and exported to C code with the automatic code generation. The dynamic RRT-Star $\mathcal{T}$ is created with

$$q_{start} = \begin{pmatrix} 0.2 \\ 2 \\ 0 \end{pmatrix}, \; \tilde{q}_{goal} = \begin{pmatrix} 2.6 \\ 2 \end{pmatrix}, \; \tilde{q}_{Obst} = \left[\begin{pmatrix} 1 \\ 1 \end{pmatrix} \begin{pmatrix} 1.6 \\ 3 \end{pmatrix} \right], \; d_{Safety} = 0.4, \; N_{max} = 3000. \quad (3.11)$$

q_{start} denotes the initial position of the robot, $\tilde{q}_{goal}$ is the goal position, which can be reached with any heading angle, d_{Safety} is the minimal safety distance of a robot to an obstacle which renders a node as valid and N_{max} is the total number of random points that are checked during the algorithm. Without loss of generality, only rectangular obstacles $\tilde{q}_{Obst} = \left[\tilde{q}_1 \; \tilde{q}_2\right]$ are considered, where $\tilde{q}_1$ is the position of one corner and $\tilde{q}_2$ of the other. The remaining corners are created by interchanging the respective $x-$ or $y-$coordinates. Because of the discretization, the other values like d_{min}, d_{max} or the goal region no longer have to be taken into account.



(a) Optimal path by dynamic RRT-Star algorithm.

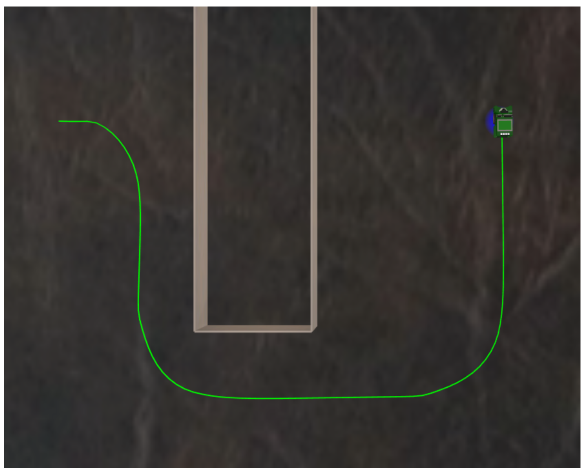

(b) Robot follows the generated optimal path.

Figure 3.4: Simulation of the dynamic RRT-Star Algorithm.

The dynamic RRT-Star algorithm results in the optimal path of fig. 3.6b, which is combined with the controller to drive a robot to a goal configuration, while the known obstacle is avoided. The sequence of nodes of this path and the inputs of the corresponding primitives are generating a time depending trajectory. This trajectory is used as a virtual leader for the robot to determine the desired configurations q_d. To achieve smooth transitions and no abrupt changes, the path is interpolated in between two nodes. Figure 3.4b shows the performance of the robot, while the optimal path is followed with the same control parameters as used in section 3.1.2. It is easily verifiable that the robot is able to follow the intended trajectory. Although only a spatial representation of the results is chosen, it can be concluded that the robot reaches $\tilde{q}_{goal}$ with the optimal time of 38s. With this extension, the robot is able to approach a goal configuration with a smooth trajectory in optimal time, while obstacles are avoided.

3.3 Obstacle Avoidance

The just mentioned dynamic RRT-Star algorithm considers, that the optimal path does not hit any obstacle. However, the topic of collision avoidance has to be readdressed, as the algorithm can only incorporate known obstacles. If the location of an obstacle varies, or is not known at all, it can not be guaranteed that there is no collision. Therefore an extension for the leader-follower tracking controller is proposed, which does not require knowledge of the location of the obstacle.

3.3.1 Control Algorithm

In order to detect any obstacle close to the robot, the leader-follower tracking controller from section 3.1 is extended to use the on-board position sensitive devices. The sensor readings are now evaluated in order to manipulate the desired coordinates of the controller to avoid any approaching obstacle.
Inspired by [26], a nearby obstacle is detected by the measurements of the sensors PSD_x. x indicates whether the sensor is mounted on the left, front or right side of the robot and determines on which side the obstacle is located. If a value of a PSD_x reading falls below a limit d_{Coll}, the path of the robot must be changed, as the original trajectory probably leads to a collision. To achieve this, new desired coordinates $q^{loc}_{d,new}$ are chosen to avoid these contacts and they replace the original values of the desired coordinates q_d. If an obstacle is detected at the front, the robot turns until the obstacle is on its side. While an obstacle is located on either side, it will be followed with a safe distance until the end of the obstacle is identified.
When comparing eq. (3.4) with eq. (2.7), it can be perceived that the controller's error coordinates e^{loc} are expressed as local coordinates of the robot, while the point P with x^B_P corresponds to $x_{d,i}$ and x_i matches x^A_B. The same correlation holds for y.
Consecutively, the controller's desired coordinates q_d[1] are changed to new local coordinates $q^{loc}_{d,new}$, if the robot is close to an obstacle. It is important to notice, that q_d and $q^{loc}_{d,new}$ are expressed in different reference coordinate systems, and therefore $q^{loc}_{d,new}$ has to be transformed to the global frame to stay consistent. The cartesian error coordinates p are evaluated with $q_{d,new}$ and transformed into the local coordinates e^{loc} to calculate the control inputs.

[1]If no superscript is stated, the variable is expressed in the global world frame.

Table 3.1: Different cases for collision avoidance.

case	x_d^{loc}	y_d^{loc}	θ_d^{loc}
$1 : PSD_L < d_{Coll}$	$\frac{PSD_L}{d_{Coll}} d_{x,L}$	$-(d_{Coll} - PSD_R)K_{PSD,L}$	$atan2(y_d^{loc}, x_d^{loc})$
$2 : PSD_F < d_{Coll}$	$(PSD_F - d_{Tol})K_{PSD,F}$	$\pm d_{y,front}$	$atan2(y_d^{loc}, x_d^{loc})$
$3 : PSD_R < d_{Coll}$	$\frac{PSD_R}{d_{Coll}} d_{x,R}$	$(d_{Coll} - PSD_R)K_{PSD,R}$	$atan2(y_d^{loc}, x_d^{loc})$

However, all of these steps can be omitted, when any of the cases in table 3.1 holds true. In such situation, e^{loc} is directly replaced with the new desired local position $x_{d,new}^{loc}$, $y_{d,new}^{loc}$ and $\theta_{d,new}^{loc}$ from table 3.1.
In case 1 or 3, the closer the obstacle is located to the side and $PSD_{L/R} \to 0$, the smaller x_d^{loc} gets while increasing y_d^{loc}. Thus, the robot prioritizes turning over driving straight, based on the proximity to the obstacle. y_d^{loc} is controlled by a proportional controller (P-controller) with the proportional gain K_{PSD}. The sign indicates the side to which the robot is turning to. This controller allows the robot to follow the obstacle with the designated distance d_{Coll}. If the robot is facing away from the obstacle, it is controlled to turn towards it and stay close in order to drive along the obstacle. Otherwise, the robot would assume that it has passed the obstacle and return to its original q_d. This would result in a zig-zag like path along the obstacle, if the desired configuration is located on the other side. This is avoided when the P-controller is used.
x_d^{loc} for case 2 is also controlled by a P-controller. This implies that even if an obstacle is detected at the front, the robot is slowly driving forwards until a tolerance threshold d_{Tol} is reached. If the threshold is obtained, x_d^{loc} is zero and therefore the controlled linear velocity v is also zero. Meanwhile, the robot is turning towards y_d^{loc}. To guarantee that the robot does not hit the obstacle at the front, the robot drives backwards, if PSD_F falls below the threshold since $x_d^{loc} < 0$. The constant $d_{y,front}$ for y_d^{loc} turns the robot in the direction which is not obstructed by an obstacle. Checking whether $PSD_L \geq PSD_R$ holds, yields the sign of y_d^{loc} and the robot turns to the corresponding side.
For all cases, θ_d^{loc} is determined by the 2-argument arctangent to calculate the desired steering angle correctly to steer towards $q_{d,new}$. If $PSD > d_{end}$, the end of the obstacle is detected and the robot continues to drive straight for t_{end} in order to ensure, that the obstacle is passed and an imminent collision with a possible corner is avoided. Afterwards, the original q_d becomes valid again and the robot resumes to steer towards its desired configuration.
In contrast to the previous introduced RRT algorithm, this extension does not require any knowledge of the location of an obstacle and only relies on the local information of the PSD sensors. Hence this is a valuable expansion to the leader-follower tracking controller.

3.3.2 Simulation

This section evaluates the functionality of the leader-follower tracking controller with collision avoidance. A robot is driving towards a goal, while two obstacles are avoided. The location of these obstacles are not known to the robot.

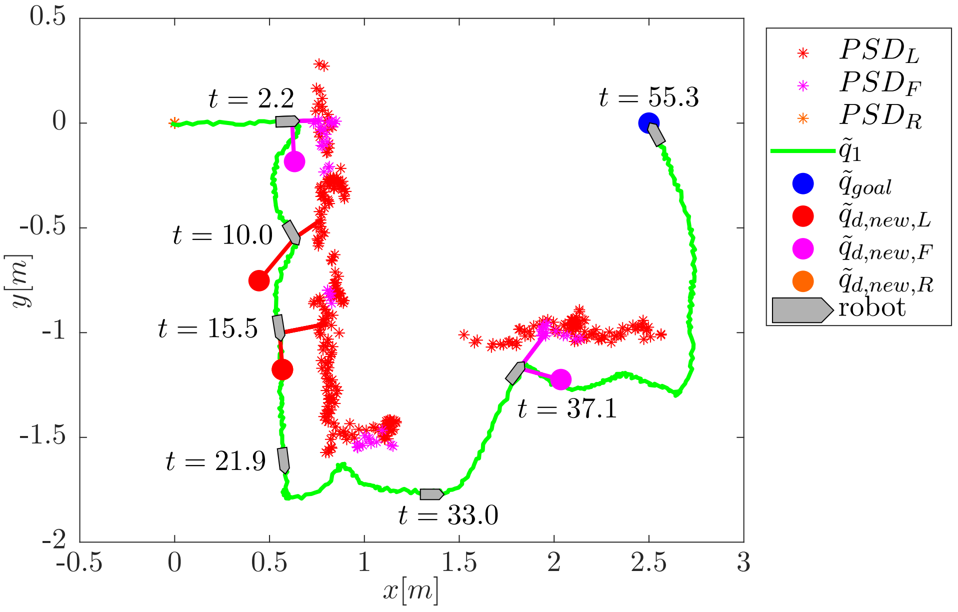
The world is setup with

$$\begin{gathered} q_1(0) = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix}, q_{d,1} = \begin{pmatrix} 2,5 \\ 0 \\ atan2(p_{y,1}, p_{x,1}) \end{pmatrix}, \\ \tilde{q}_{Obst,1} = \left[\begin{pmatrix} 0.8 \\ -1 \end{pmatrix} \begin{pmatrix} 1.2 \\ 0.5 \end{pmatrix} \right] \text{ and } \tilde{q}_{Obst,2} = \left[\begin{pmatrix} 1.5 \\ -0.5 \end{pmatrix} \begin{pmatrix} 2.5 \\ -0.3 \end{pmatrix} \right]. \end{gathered} \tag{3.12}$$

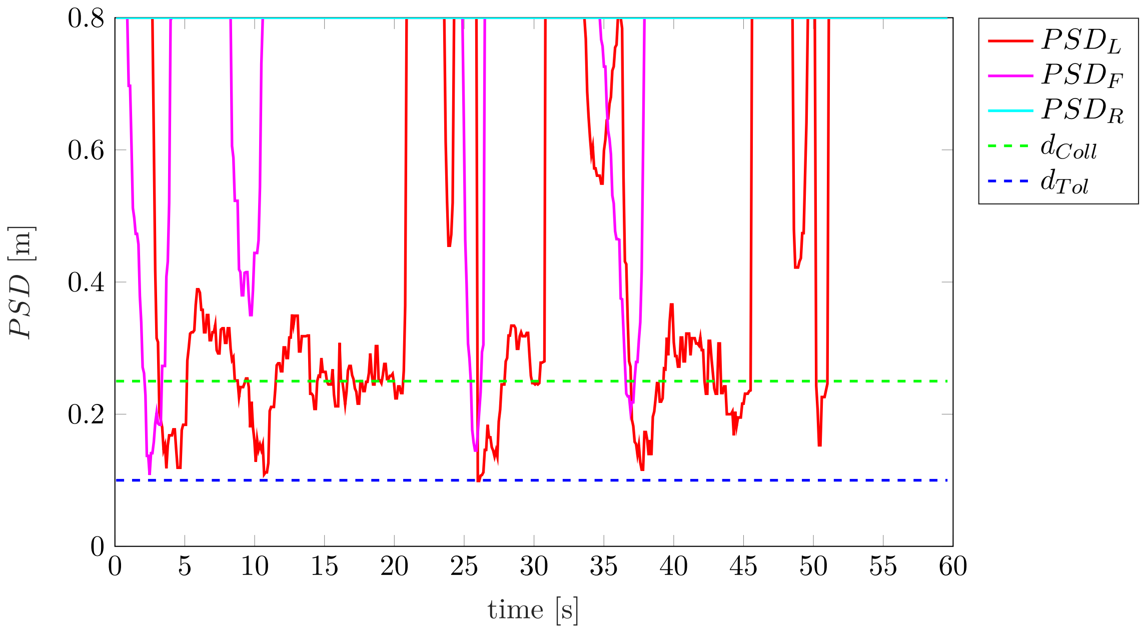
These locations are chosen, such that the first obstacle is positioned at the front of the robot, whereas the second obstacle is approached lateral. The parameters for the collision avoidance extension are

$$\begin{gathered} d_{Coll} = 0.25,\ d_{Tol} = 0.1,\ K_{PSD,L/R} = 1.5,\ K_{PSD,F} = 0.3, \\ d_{x,L/R} = 0.3,\ d_{y,front} = 0.5,\ d_{end} = 0.5 \text{ and } t_{end} = 2 \end{gathered} \tag{3.13}$$

The variables correspond to table 3.1. If the PSD sensor facing the obstacle exceeds d_{end}, the robot assumes to have passed the obstacle and drives straight for the duration of t_{end} before it resumes to drive towards the original q_d. The parameter must lead to avoid the obstacles for both the frontal and lateral case. The leader-follower tracking controller is used as a point-to-point position controller, where all of the control parameters stay unchanged to section 3.1.2. Both position noise and noise for the PSD sensors are considered.
The scenario is illustrated in fig. 3.5a, where the path of the robot and the position of the detected obstacle is shown. Additionally, the pose of the robot for some selected time instances are highlighted. There, the current PSD measurement with the corresponding position of $q_{d,new}$ is depicted as well. The tip of the robot marks its front. As seen in the Figure, the robot is driving towards $\tilde{q}_{goal}$ until an obstacle is detected at the front at $t = 2.2$. At first, the robot continues to drive forwards while turning. Since the robot continues to approach the wall, it slows down and the turning becomes more dominant. As soon as the obstacle is on the side of the robot, it is controlled to follow the wall while staying close to the obstacle. The effect of the P-controller for y_d^{loc} can be seen at $t = 10.0$ and $t = 15.5$, where the desired location is placed based on the proximity to the obstacle. At $t = 21.9$ the edge of the obstacle is detected and the robot continues to drive straight for t_{end}, only to encounter the next side of the obstacle shortly after. This new side is followed until $t = 33.0$. After t_{end} the robot resumes to drive towards $\tilde{q}_{goal}$. At $t = 37.1$ the second obstacle is detected, which is approached lateral by the robot. Again, an avoidance maneuver is performed to drive around the obstacle. Eventually the goal configuration is reached at $t = 55.3$ without any obstacle being hit. This conclusion is confirmed by fig. 3.5b, where the measurements of the PSD sensors are illustrated. None of the values indicate a collision and the tolerance threshold d_{Tol} is never undercut. The effect of controlling PSD_L to d_{Coll} for case 1 and thereby the robot following the wall can be perceived as well for the interval of $t = [5, 20]s$ and to some extent during $t = [37, 45]s$.
This simulation leads to the conclusion, that this extension prevents the robot to hit any obstacle. Not only a collision is avoided, but the robot is also navigating around the obstacle by following the wall, until an edge is detected. Afterwards, the robot resumes to drive towards its original goal configuration, although it is possible to directly avoid a new wall, if the target is still located at the direction of the obstacle.



(a) State information of the robot with the position of the detected obstacles.



(b) Measurements of the PSD sensors, while obstacles are avoided.

Figure 3.5: Results of the leader-follower tracking controller with collision avoidance extension.

3.4 Combined Implementation

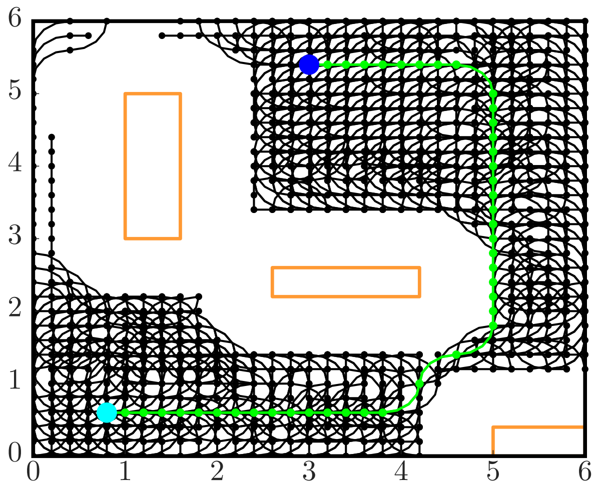
This section combines all of the previously introduced control schemes in order to navigate two robots through an extended world setup. On the one hand, the lead robot uses the leader-follower tracking controller to drive along the optimal path created by the dynamic RRT-Star algorithm. On the other hand, the follower uses the same controller to track its leader. Both known and unknown obstacles will be avoided while the robots are approaching their goal position.
There are four obstacles in the extended world setup, where three of them are known to the robots. The path of the leader is generated by a dynamic RRT-Star algorithm, while the known obstacles are incorporated. The world is designed such that at some point the robots will encounter the unknown fourth obstacle, which will be avoided with the local PSD sensors. The parameters of the RRT and the controller for this scenario are

$$q_{1,start} = \begin{pmatrix} 0.8 \\ 0.6 \\ 0 \end{pmatrix},\ \tilde{q}_{1,goal} = \begin{pmatrix} 2.6 \\ 5.4 \end{pmatrix},\ q_{2,start} = \begin{pmatrix} 0.3 \\ 0.6 \\ 0 \end{pmatrix},\ d_{d,x,2} = 0,\ d_{d,y,2} = -0.4, \\ d_{Coll} = 0.25,\ d_{Safety} = 0.8 \tag{3.14}$$

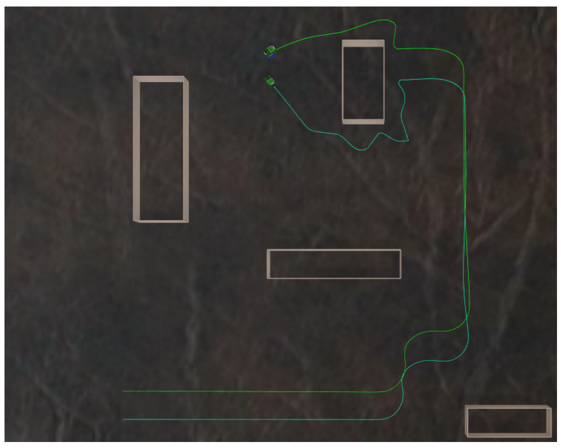
and the world is created with

$$\tilde{q}^{known}_{Obst,1} = \left[\begin{pmatrix} 1 \\ 3 \end{pmatrix} \begin{pmatrix} 1.6 \\ 5 \end{pmatrix}\right],\ \tilde{q}^{known}_{Obst,2} = \left[\begin{pmatrix} 2.6 \\ 2.2 \end{pmatrix} \begin{pmatrix} 4.2 \\ 2.6 \end{pmatrix}\right], \\ \tilde{q}^{known}_{Obst,3} = \left[\begin{pmatrix} 5 \\ 0 \end{pmatrix} \begin{pmatrix} 6 \\ 0.4 \end{pmatrix}\right],\ \tilde{q}^{unknown}_{Obst,4} = \left[\begin{pmatrix} 3.5 \\ 3.8 \end{pmatrix} \begin{pmatrix} 4 \\ 6.5 \end{pmatrix}\right]. \tag{3.15}$$

To result in a reasonable trajectory for the follower with the collision avoidance controller, the robot must be controlled to keep an appropriate distance to the leader. No collision should be detected while the robot is following adequately and therefore the displacements must be chosen such that $|d_{d,2}| > d_{Coll}$ holds. During the RRT algorithm, the minimal safety distance d_{Safety} of a node to an obstacle, which defines the node as valid, is increased. Thereby the planning algorithm can consider not only the leader, but also incorporate that the desired optimal path of the second robot is not obstructed by an obstacle. Hence, the value is increased to validate $|d_{d,2}| + d_{Coll} < d_{Safety}$. In order to avoid following across an obstacle, the desired location within the optimal path, instead of the current position of the leader, is transmitted to the follower via the radio. Meanwhile, the follower robot i has to set v_{i-1} and ω_{i-1} locally to zero during the avoidance maneuver, so the computed inputs do not get corrupted. Additionally, θ_d has to be changed to the 2-argument arctangent, while resuming to the original trajectory, after the robot has evaded an obstacle. Unless mentioned previously, all other parameters for the controller, the RRT algorithm or the collision avoidance extension are unchanged.
The setup of the extended world, the optimal path by the dynamic RRT, and the traveled path of both robots are visualized in fig. 3.6. The optimal time predicted by the RRT is $t_{opt} = 71s$. Figure 3.7 shows the results of the simulation, where the coordinates, desired coordinates and the PSD sensors for the leader and the follower are depicted. For the first 60s, the robots are navigating through the world, as intended by the RRT. The robot follows the leader with the desired offset, without the need to avoid either an obstacle or the leader. Based on the orientation, either the left, front or right PSD sensor detects the position of the



(a) Optimal path by dynamic RRT-Star algorithm.

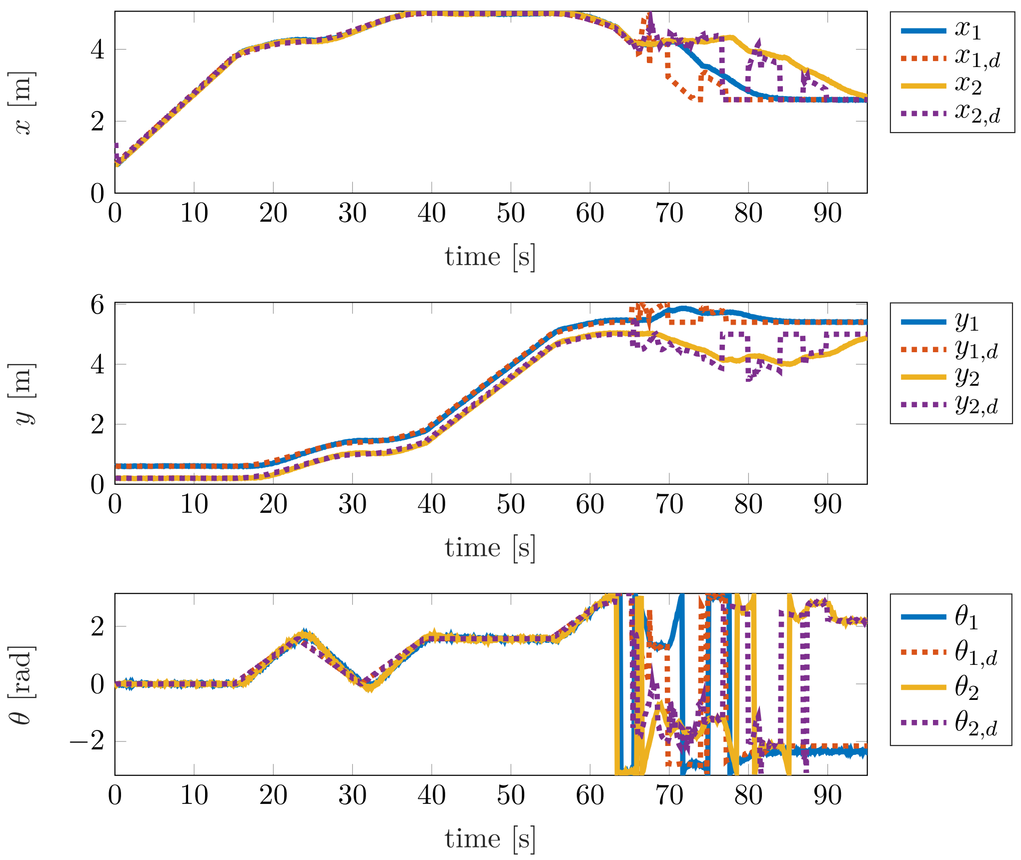


(b) Optimal path by dynamic RRT-Star algorithm.

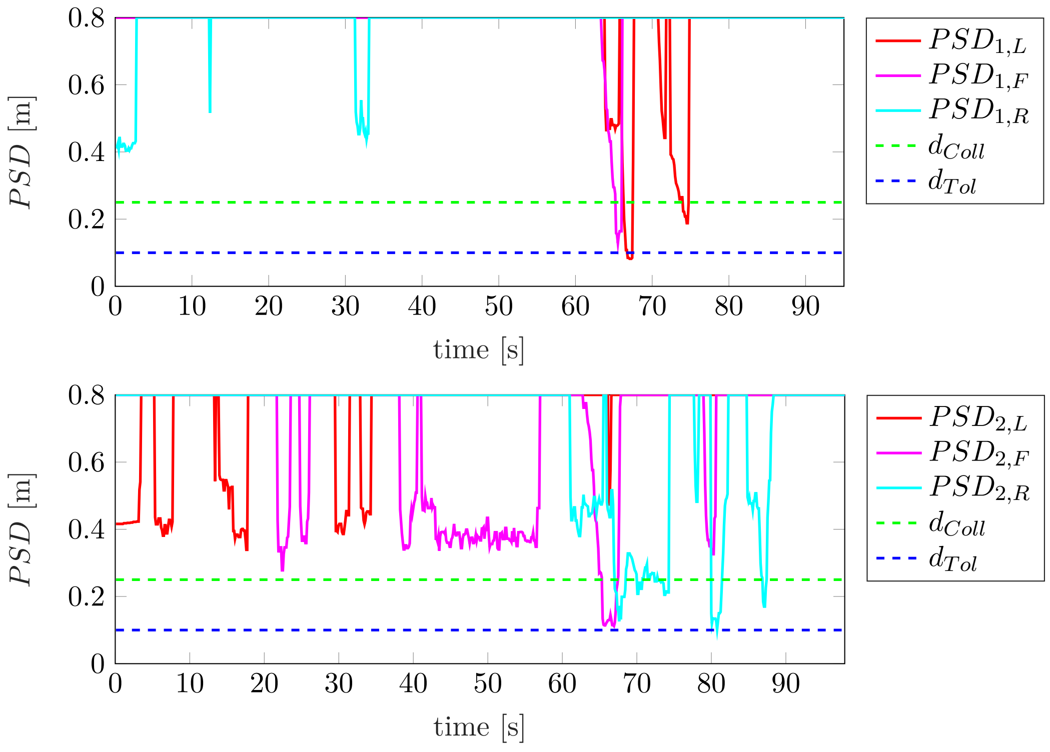
Figure 3.6: Spatial representation of the extended world simulation.

leader. If the sensors are aligned suitably, they indicate the desired displacement to the leader $d_d = 0.4$. At $t = 65s$, the unknown obstacle is encountered. As the leader detects the position of its follower on the left side, the condition $PSD_{L,1} < PSD_{R,1}$ holds true. Therefore, the leader is turning to the right side of the obstacle. The follower experiences the reverse case and hence drives around the left side of the obstacle. The desired coordinates q_d are changing continuously[2] based on the proximity to the obstacle, while the robots are driving around the obstacle. After 90s, both robots have passed the obstacle and continue to drive towards their goal position, which is reached after 95s.
Overall, the robots are navigating through the extended world and reach their target position without an obstacle being hit. As the fourth, unknown obstacle is blocking the optimal path, the robot has not reached its target within the predicted time. However, with the collision avoidance extension, the robots are able to navigate around the obstacle and eventually reach the target position.

[2]The discontinuous course of the steering angle θ is explained by its restriction to $[-\pi, \pi]$.



(a) State information q_2, desired coordinates $q_{d,2}$ of the follower and q_1 of the leader.



(b) PSD readings of the leader robot 1 and follower robot 2 in the complex world.

Figure 3.7: Sensor readings during the simulation of the extended world.

Chapter 4

Experimental Results and Discussion

This chapter will discuss the implementation of the previous algorithms. First, the robots used for the experiments are presented and some remarks of their functionality are highlighted. Secondly, the previously simulated scenarios are repeated with real experiments. During the analysis, problems and improvements are highlighted. In the end, the robots are used to navigate through an extended world without any collisions

4.1 EyeBot Robots

The EyeBot robots are developed as part of the EyeBot project, introduced in section 2.1.2. The operating system of an EyeBot robot is run by a Raspberry Pi 3, while an Arduino microcontroller is processing the low level functions, sensors and motors. Being a differentially driven robot, the velocity of every wheel is set independently, however with the transformation of eq. (2.1) it is also possible to directly set the linear and angular velocity of the robot. An internal PID controller is controlling the effective velocity to the desired one. The displacements of both wheels are measured by encoders to calculate the position for each cycle of the IO board. The position and velocity of the robot are calculated by incrementing these displacements regarding their geometric relation. The cycle time of a robot corresponds to the time that is required to execute all functions that are looped, like reading the sensors, calculating the control inputs and sending a message.
Every EyeBot robot is equipped with three PSD sensors to detect the distance towards an object. They are located at the front, left and right side of the robot. In order to reduce the effect of the measurement noise, their effective value is created by a mid-value selection of the raw readings. Therefore it is required to save the last two raw sensor readings. For every time instance, the two saved measurements and the current sensor reading are sorted by size. The middle value of every pair of three becomes the current effective PSD reading. As the last two raw values are stored, this method results in an averaged signal for every time instance, without delaying the reading of the sensor.
By connecting the EyeBot robots to the same Wifi, they can communicate with each other. The IP address of a robot is used to uniquely identify every vehicle and therefore it is possible to exchange messages with information between robots. The continuous communication between robots can overfill the communication buffer, if the robot, which continuously sends a message, has a faster cycle time than the other. To avoid this, the sending robot always waits until the receiver is signalizing when it is ready to receive a new message. Since the messages are read on a first-in-first-out basis, receiving a delayed message is prevented, and every message corresponds to the real-time response and is not delayed.

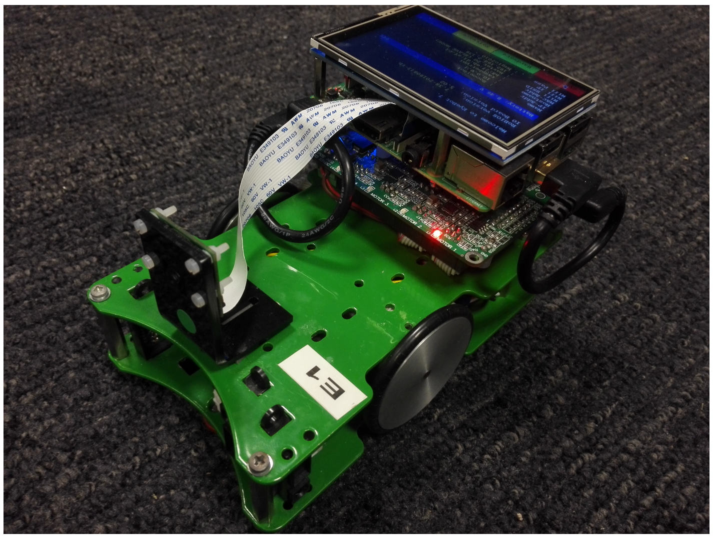


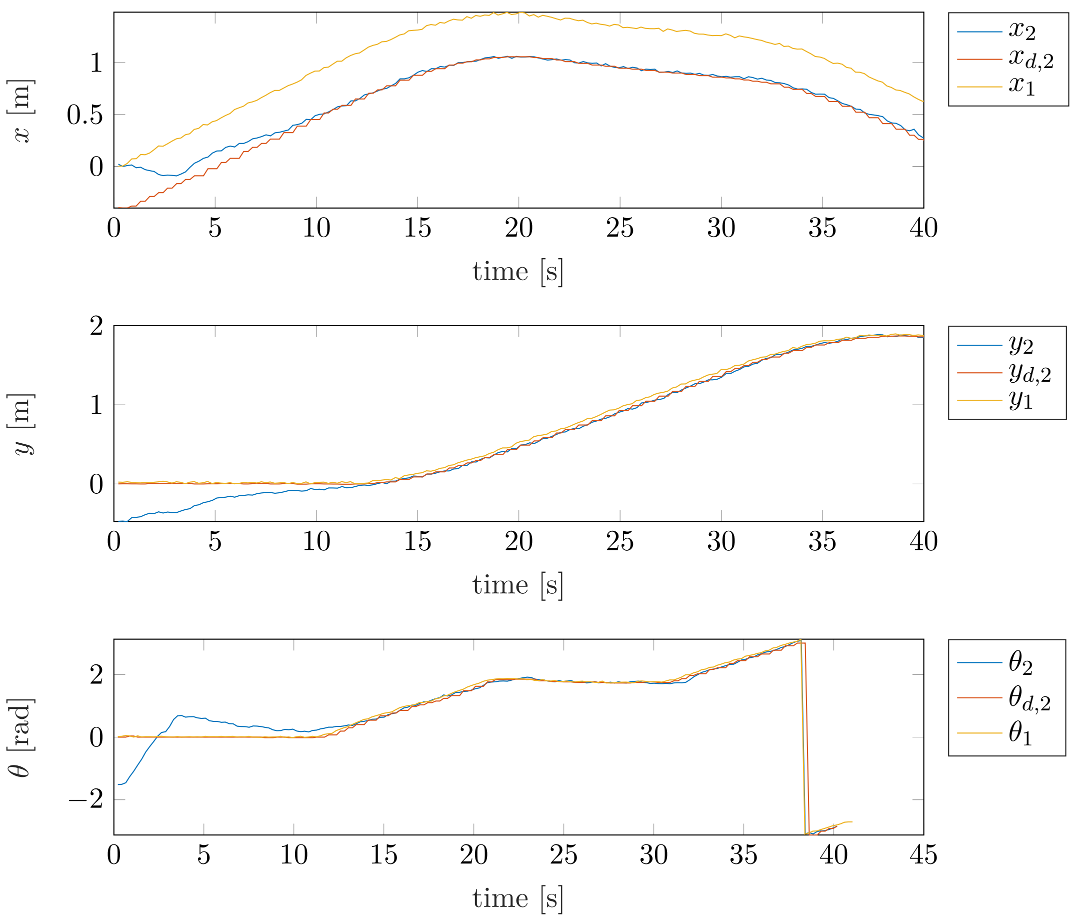
Figure 4.1: Realization of the EyeBot robot.

4.2 Leader-Follower Tracking controller

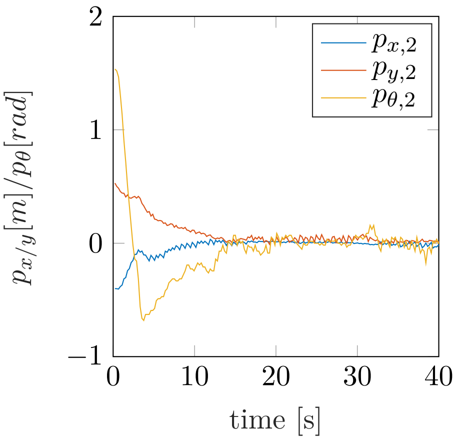
In order to evaluate the functionality of the leader-follower tracking controller with the EyeBot robots, the scenario of section 3.1.2 is replicated, where a lead robot is repeatedly changing from driving straight to driving a curve. This leader is tracked by a follower with the control parameters

$$d_{x,2} = -0.5, d_{y,2} = 0 \quad \text{and} \quad c_a = 0.5, c_b = 0.5, c_c = 1. \tag{4.1}$$

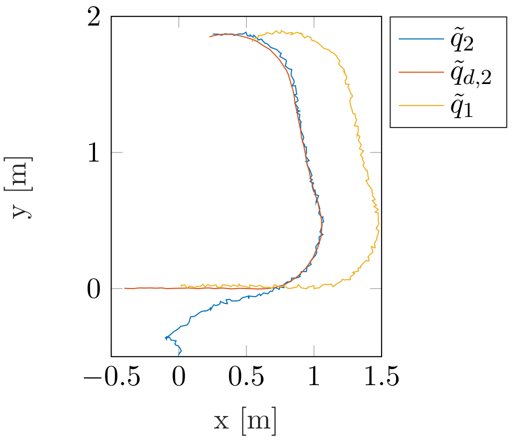
The control parameters are chosen such that $c_c >> c_b$ holds. This leads to the effect, that the controller emphasizes the stabilization of y over a correct steering angle θ. If the continuous tracking of the $x-$ and $y-$coordinate is correct, then, without loss of generality, the steering angle of the robot is also correct. Therefore the controller achieves enhanced performance in this case.
The result of this experiment with the EyeBots is visualized in fig. 4.2. It can be seen, that the follower attains the desired configuration and follows the leader with the chosen offset. Just like in simulation, the controller's good performance is indicated by the error coordinates p_2. Analyzing fig. 4.2a leads to the conclusion that the messages received by the follower correspond to the up-to-date position of the leader, as the time delay between $y_{d,2}$ and y_1 is neglectable small. The nature of receiving the desired values discretely is related to the fact that processing the physical signals from the Raspberry to the IO Board back and forth, like reading the sensors and setting the motors, takes most of the time for every cycle. If the leader trajectory does not suffice the system dynamics of a unicycle, it is important to use the two argument arctangent to calculate $\theta_d = atan2(p_{y,i-1}, p_{x,i-1})$. Otherwise, further path planning to reach a position with a desired orientation, e.g. with Hermite splines [31], can be implemented in future research.



(a) State information q_2, desired coordinates $q_{d,2}$ of the follower and q_1 of the leader.



(b) Error coordinates p of the follower.



(c) Spatial representation of the traveled path.

Figure 4.2: Results of the leader-follower tracking controller with two EyeBoot robots.

4.3 Path Planning

Like in the previous section, the scenario of the simulation of the dynamic RRT-Star is repeated with the same parameters. The EyeBot robot solves locally the RRT-Star algorithm for the optimal path, which is then used to be followed with the leader-follower tracking controller. The resulting trajectory of the EyeBot robot is visualized in fig. 4.3. As expected, the optimal path and therefore the trajectory of the desired coordinates q_d is similar to the simulation, as no parameters were changed. The total traveled path of the robot indicates the successful tracking behavior. The path verifies that there was no collision, as this would result in a stop of the robot. Thus, the objective of the dynamic RRT-star algorithm is accomplished.
The current implementation only considers the system dynamics of a unicycle robot. Therefore, the inputs of the linear velocity v and angular velocity ω are not restricted to be continuous. This means that, the input signals can jump, when different motion primitives are used. This can be a problem in particular for the correct θ, if ω needs to ramp up or down to reach the desired value, due to physical restrictions. However, as the path consists not only of v and ω but also of the position of the nodes, this effect only influences the high-end precision of the robot. This effect can also be seen, if the results of tracking the real robot in section 4.2 and following the dynamic RRT are compared. If the leader is a real robot, then the transmitted inputs correspond to the actual values, instead of the theoretical ones, and therefore the follower achieves enhanced performance in this case. Although reasonable results are achieved with the current implementation of the dynamic RRT, the high-end accuracy can be improved in future research by addressing these issues.
The main drawback of the dynamic RRT-Star is currently the limited number of motion primitives and therefore the coarse discretization of the nodes. As already discussed during the design, it is easily verifiable that the optimal path is only optimal with respect to the allowed primitives and the chosen time values. A path which favors transversal movements will lead to an overall shorter total distance. If a primitive allows to steer towards a ±45° angle, this option allows the optimal path to converge to the true optimum remarkably. Besides, additional trajectory optimization, like increasing the speed while driving along multiple straight segments, can be considered in the future.

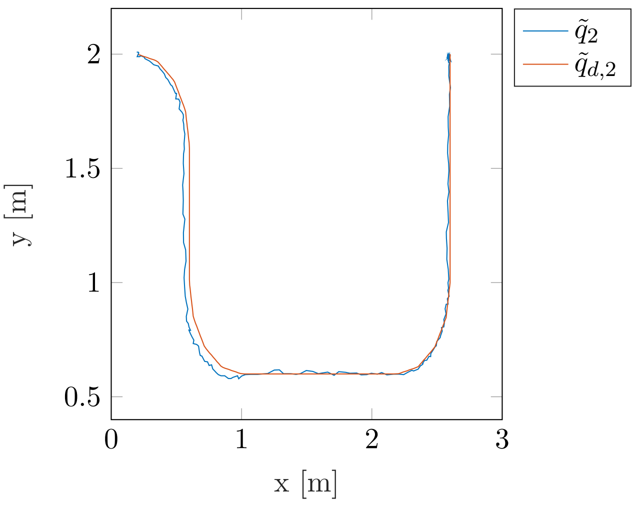


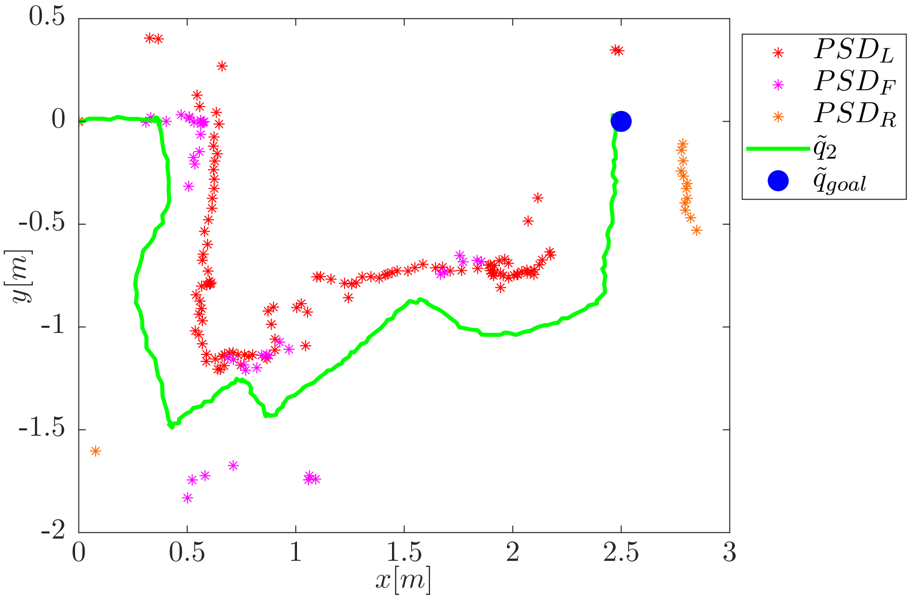
Figure 4.3: Spatial path of the EyeBot following the optimal path of the dynamic RRT-Star algorithm.

4.4 Collision Avoidance

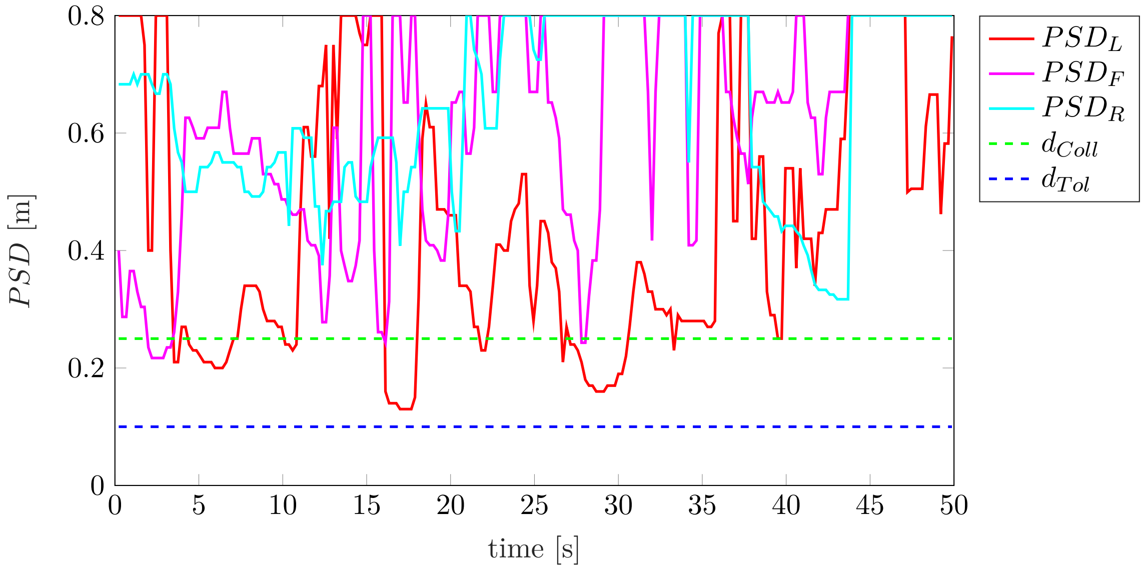
This section presents the performance of an EyeBot robot avoiding collisions with the PSD sensors, while the leader-follower tracking controller is used as a point-to-point position controller.
The EyeBot robot is using the collision parameters

$$\begin{gathered} d_{Coll} = 0.3,\ d_{Tol} = 0.1,\ K_{PSD,L/R} = 1.5,\ K_{PSD,F} = 0.75, \\ d_{x,L/R} = 0.4,\ d_{y,front} = 0.8,\ d_{end} = 0.5 \text{ and } t_{end} = 2. \end{gathered} \tag{4.2}$$

Every other parameter is similar to the previous values. Primarily, it can be noticed during the analysis of the results in fig. 4.4, that the robot reaches the target, without hitting an obstacle. Although the PSD readings are significantly impacted by noise, the path of the EyeBot is reasonably accurate. The PSD sensors are capable of measuring the distance of an object within $0.8m$, but due to noise, these values are not very reliable. However, most of the readings below $0.4m$ correspond to a reasonably correct distance towards an object. Therefore d_{Coll} should be chosen below this threshold. The effect of the noise can specifically seen at PSD_R, as there is no obstacle on the right-hand side over the course of the experiment.
The value t_{end} is of major importance, which determines how long the robot resumes driving straight, after the edge of the obstacle has been passed. If the goal is still located behind the obstacle, the robot must not hit the new side of the obstacle, while turning towards $\tilde{q}_{goal}$. In rare cases, detecting the end of an obstacle with $PSD_x > d_{end}$ is a concern. As the robot is following the obstacle and therefore controlled to stay at $PSD_{L/R} = d_{Coll}$, the robot turns towards the side of the obstacle, if $PSD_{L/R} > d_{Coll}$ holds. Consequently, the robot might be turning towards the new side of the obstacle and miss the detection of the edge. This leads to another drawback of this approach, as the robot is always following the wall until an edge has been detected. The robot does not resume to drive towards the target, even if the obstacle is located to the other side. Aside from that, the collision avoidance extension can be upgraded, if additional sensors are attached to the EyeBots. Specifically mounting sensors at the front left and front right position of the robot will help to identify the direction of an obstacle, which is approached laterally. Thus, objects at $\pm 45°$ in addition to the current $0°$ and $\pm 90°$ can be measured.
Although the sensor readings are very noisy, this extension allows the EyeBots to prevent an imminent collision, follow an obstacle until an edge is detected and eventually continue to drive towards the target configuration. Therefore, the collision avoidance extension can also be used with the real EyeBot robots.



(a) State information of the robot with the position of the detected obstacles.



(b) Measurements of the PSD sensors.

Figure 4.4: Results of the EyeBot robots using the collision avoidance extension.

4.5 Combined Implementation

As all of the previous control schemes are successfully applied to the EyeBot robots, this section investigates their combination in order to navigate through the extended world of section 3.4. Like in the simulation, the location of the obstacles $\tilde{q}_{Obst,1-3}$ are known to the robots and the fourth obstacle $\tilde{q}_{Obst,4}$ is unknown. Robot 2 is controlled to follow the lead robot 1 with an offset in $y-$coordinates. The path of the leader is generated with the dynamic RRT-Star algorithm, such that both the leader and the follower do not come close to a known obstacle. The required adaptation for the parameters was already discussed in section 3.4 and as the previous, individual control schemes are already experimentally evaluated, no control parameter has to be changed.
A visualization of the results can be seen in fig. 4.5 and fig. 4.6. Figure 4.5 shows the total traveled path of both robots, with their desired position and PSD readings. For the sake of clarity, it is abstained to display any PSD reading above the threshold $d_{Coll} = 0.25$, and q_d after the unknown obstacle is detected at $t = 65s$. The figure illustrates, that the goal configuration is reached by the leader and the follower. The state variables q in fig. 4.6a indicate, that the leader is driving successfully along the optimal path, generated by the dynamic RRT-Star, for the first 65s. The follower is tracking its leader with the desired offset in $y-$coordinates. After the unknown obstacle has been encountered by both robots, the collision avoidance extension is changing the desired configuration for the controller. After the obstacle is passed, the leader reaches its target at $t = 83s$ and the follower eventually at at $t = 95s$. In contrast to the simulation, the PSD sensors do not give as much insight into the course of the robots. However, most importantly fig. 4.6b confirms, that no robot collides with any obstacle throughout the course of the experiment. Secondly, the figure indicates that, due to the path planning, no obstacle is detected until $t = 65s$.

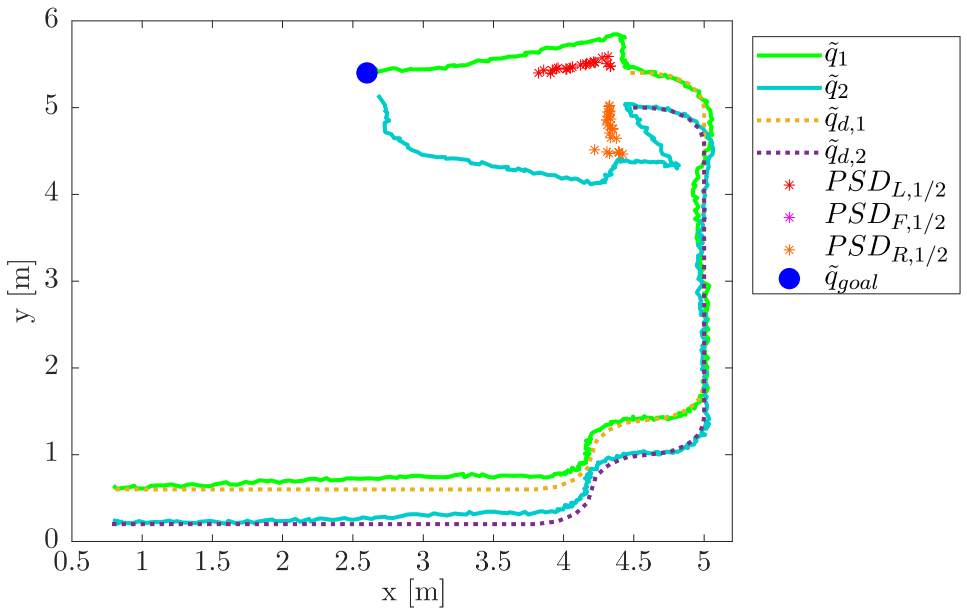
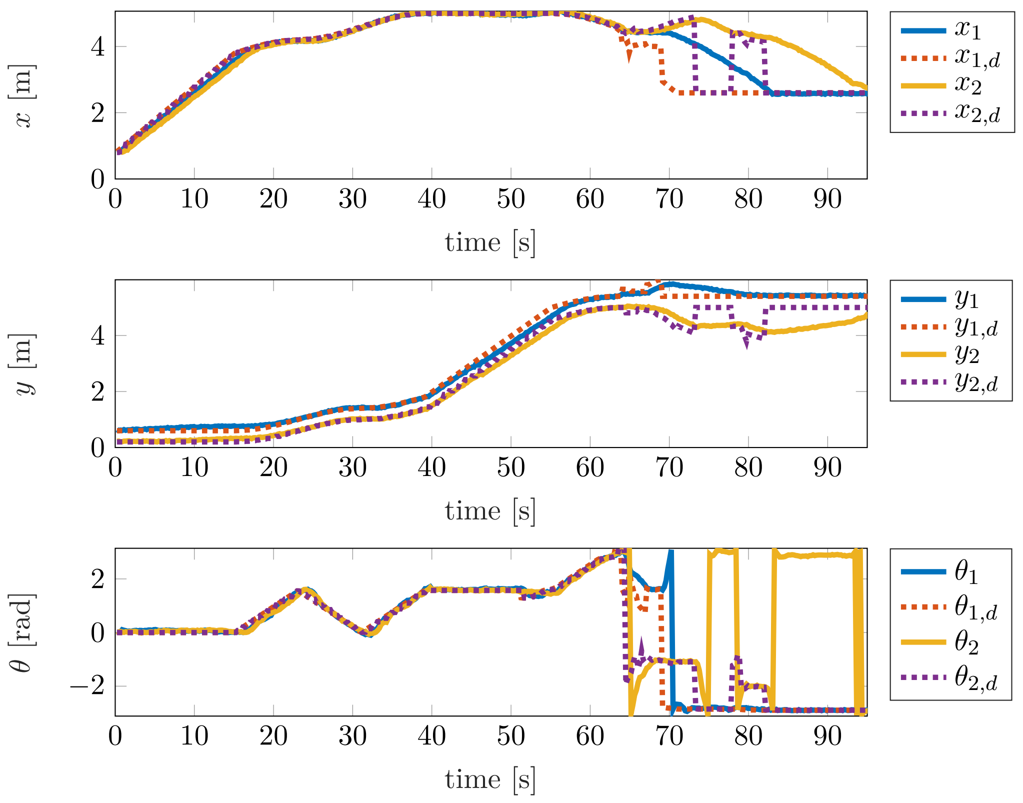
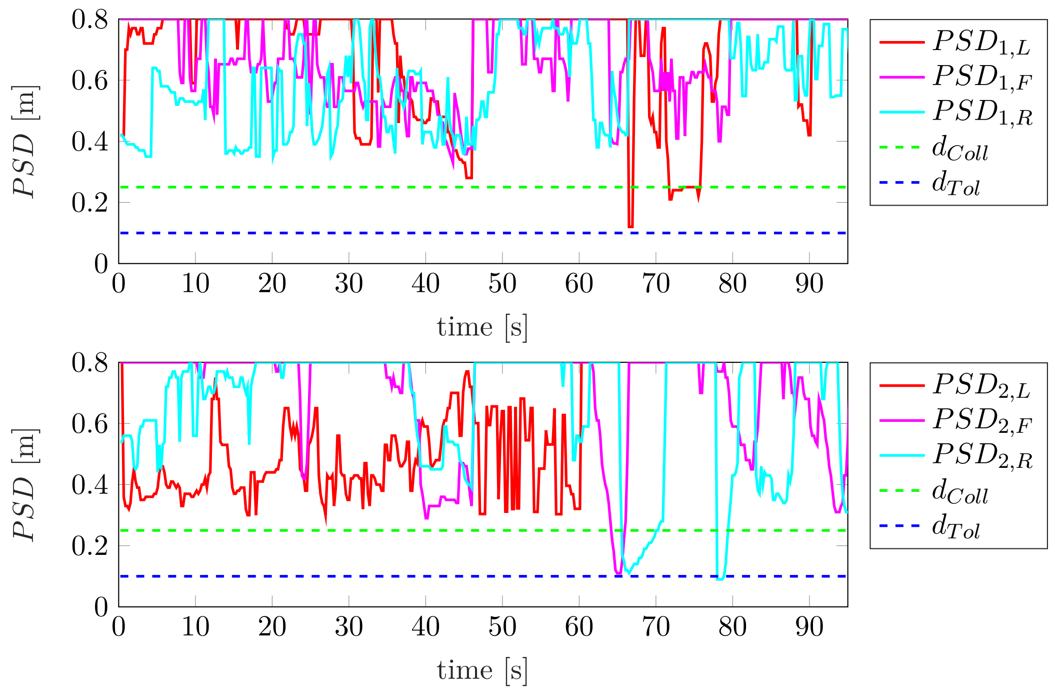


Figure 4.5: Spatial representation of the EyeBot leader $\tilde{q}_1$ and the follower $\tilde{q}_2$, with their desired coordinates $\tilde{q}_d$ and PSD readings, in the extended world.



(a) State information q_2, desired coordinates $q_{d,2}$ of the follower and q_1 of the leader.



(b) PSD readings of the leader robot 1 and follower robot 2 in the extended world.

Figure 4.6: Sensor readings of the EyeBot robots during the simulation of the extended world.

If the purpose of the robots is to explore an environment with little to no knowledge of the location, an algorithm, which guarantees to reach the target without getting stuck in a local dead end, should be used. An analysis of navigation algorithms in unknown environments was conducted in [32] and for example a distbug algorithm [33] can be employed. The dynamic RRT-Star algorithm is only solved during initialization and is not suitable to find an optimal path during operation. In contrast, a Model Predictive Control approach [34] solves an optimization problem online during operation of the robot. This possibility allows to enhance the performance of the EyeBot robots and can be considered in future studies.
If the simulation results of all the different scenarios from chapter 3 are compared to their corresponding experiments, it is noticeable, that the simulation hardly differs from the experimental evaluation. Thus, the software EyeSim is very advantageous for the development of differentially driven robots. As the PSD sensors and measurement noise are already considered in simulation, a realistic representation of the EyeBot robots facilitates the implementation on the real robots.
Overall, a combination of control schemes is found to navigate multiple EyeBot robots through an extended world setup. As all of the experiments are conducted with the same control parameters, the robots can navigate through any arbitrary world setup by simply adjusting the initial setup and desired configuration. If the position of the obstacles is known to the robots, they reach their target along a time optimal trajectory, without encountering any obstacle. However, if the knowledge is only partially valid, the robots are still avoiding the obstacle and continue to drive towards the goal configuration, after the obstacle has been passed.

Chapter 5

Summary

In this thesis, a combination of control schemes is found to navigate EyeBot robots to a goal position, while obstacles are avoided.
This is achieved by a leader-follower tracking controller, which operates a follower to track its leader with a desired position displacement. The leader is using the same control scheme to drive along a trajectory, which is generated with an optimal path planning algorithm, the dynamic RRT-Star. This path is time optimal and during the design it is considered, that not only the leader itself, but also the follower keep a safe distance to any known obstacle. However, this method requires knowledge of the location of the obstacles. As this information is not always available, a variation of the leader-follower tracking controller was proposed, in order to detect and avoid any additional, unknown obstacles. Using the on-board PSD sensors, the desired position of the controller is adapted to prevent an imminent collision and also to drive along the side of an obstacle, until an edge is detected. After the obstacle is passed, the robot continues to drive towards the original goal. It was shown that the combination of those control schemes allows the EyeBot robots to navigate through an extended world without any collisions.
All of the control schemes were first implemented and evaluated in simulation, while noise for the position coordinates and the PSD sensors were considered. Eventually, all scenarios were successfully transferred to the EyeBot robots and executed in real experiments. Since a realistic representation of the EyeBot robots was used, the simulation results were similar to the experiment results.
Overall, the proposed combination allows the EyeBot robots to navigate through any arbitrary world setup, with full or partial knowledge of the environment.

List of Figures

List of Tables

Bibliography

[1] Yingchong Ma. „Path planning and control of non-holonomic mobile robots“. Dissertation, Laboratoire d’Automatique, Génie Informatique et Signal, Université Lille Nord, France. 2013.

[2] Martijn Rooker and Andreas Birk. „Multi-robot exploration under the constraints of wireless networking“. In: *Control Engineering Practice* 15 (Apr. 2007), pp. 435–445. DOI: `10.1016/j.conengprac.2006.08.007`.

[3] Yugang Liu and Goldie Nejat. „Robotic Urban Search and Rescue: A Survey from the Control Perspective“. In: *J. Intell. Robotics Syst.* 72.2 (Nov. 2013), pp. 147–165. ISSN: 0921-0296. DOI: `10.1007/s10846-013-9822-x`. URL: `http://dx.doi.org/10.1007/s10846-013-9822-x`.

[4] F. Solc and B. Honzik. „Modelling and control of a soccer robot“. In: *7th International Workshop on Advanced Motion Control. Proceedings (Cat. No.02TH8623).* 2002. DOI: `10.1109/AMC.2002.1026972`.

[5] Ignacio Mas. „Cluster Space Framework for Multi-Robot Formation Control“. Dissertation, Department of Mechanical Engineering, Santa Clara University, USA. 2011.

[6] Kwang-Kyo Oh, Myoung-Chul Park, and Hyo-Sung Ahn. „A survey of multi-agent formation control“. In: *Automatica* 53 (2015), pp. 424 –440. ISSN: 0005-1098. DOI: `https://doi.org/10.1016/j.automatica.2014.10.022`. URL: `http://www.sciencedirect.com/science/article/pii/S0005109814004038`.

[7] Thijs Broek, Nathan Wouw, and Henk Nijmeijer. „Formation control of unicycle mobile robots: a virtual structure approach“. In: Dec. 2009, pp. 8328–8333. DOI: `10.1109/CDC.2009.5399803`.

[8] A. K. Das, R. Fierro, V. Kumar, J. P. Ostrowski, J. Spletzer, and C. J. Taylor. „A vision-based formation control framework“. In: *IEEE Transactions on Robotics and Automation* 18.5 (2002), pp. 813–825. DOI: `10.1109/TRA.2002.803463`.

[9] A. Zhang, D. Zhou, M. Yang, and P. Yang. „Finite-Time Formation Control for Unmanned Aerial Vehicle Swarm System With Time-Delay and Input Saturation“. In: *IEEE Access* 7 (2019), pp. 5853–5864. DOI: `10.1109/ACCESS.2018.2889858`.

[10] Alessandro Gasparetto, Paolo Boscariol, Albano Lanzutti, and Renato Vidoni. „Path Planning and Trajectory Planning Algorithms: A General Overview“. In: *Mechanisms and Machine Science* 29 (Mar. 2015), pp. 3–27. DOI: `10.1007/978-3-319-14705-5_1`.

[11] P. Bhattacharya and M. L. Gavrilova. „Roadmap-Based Path Planning - Using the Voronoi Diagram for a Clearance-Based Shortest Path“. In: *IEEE Robotics Automation Magazine* 15.2 (2008), pp. 58–66. DOI: `10.1109/MRA.2008.921540`.

[12] Ahmad Abbadi and Vaclav Prenosil. „Safe path planning using cell decomposition approximation“. In: May 2015.

[13] Wojciech Kowalczyk, Maciej Michałek, and Krzysztof Kozłowski. „Trajectory tracking control with obstacle avoidance capability for unicycle-like mobile robot“. In: *Bulletin of Polish Academy of Sciences. Technical Sciences* 60 (Sept. 2012), pp. 537–546. DOI: 10.2478/v10175-012-0066-x.

[14] S. M. LaValle. *Planning Algorithms.* Available at http://planning.cs.uiuc.edu/. Cambridge, U.K.: Cambridge University Press, 2006.

[15] P. Ogren and N. E. Leonard. „A convergent dynamic window approach to obstacle avoidance“. In: *IEEE Transactions on Robotics* 21.2 (2005), pp. 188–195. DOI: 10.1109/TRO.2004.838008.

[16] Maciej Michałek and Krzysztof Kozłowski. „Vector-Field-Orientation Feedback Control Method for a Differentially Driven Vehicle“. In: *Control Systems Technology, IEEE Transactions on* 18 (Jan. 2010), pp. 45 –65.

[17] Mohamed Walid Mehrez Elrafei Mohamed Said. „Optimization Based Solutions for Control and State Estimation in Non-holonomic Mobile Robots: Stability, Distributed Control, and Relative Localization“. Dissertation, Faculty of Engineering and Applied Science Memorial University of Newfoundland, Canada. 2013.

[18] J. Chunyu, Z. Qu, E. Pollak, and M. Falash. „Reactive target-tracking control with obstacle avoidance of unicycle-type mobile robots in a dynamic environment“. In: *Proceedings of the 2010 American Control Conference.* 2010, pp. 1190–1195. DOI: 10.1109/ACC.2010.5531019.

[19] *EyeBot Website.* http://robotics.ee.uwa.edu.au/eyebot/. Accessed: 2019-10-15.

[20] Thomas Bräunl. „EyeBot: a family of autonomous mobile robots“. In: vol. 2. Feb. 1999, 645 –649 vol.2. ISBN: 0-7803-5871-6. DOI: 10.1109/ICONIP.1999.845670.

[21] N. Burleigh, J. King, and T. Bräunl. „Deep Learning for Autonomous Driving“. In: *Intl. Conference on Digital Image Computing: Techniques and Applications (DICTA)* (Dec. 2019).

[22] Steven M. Lavalle. *Rapidly-Exploring Random Trees: A New Tool for Path Planning.* Tech. rep. 1998.

[23] Sertac Karaman and Emilio Frazzoli. „Sampling-based Algorithms for Optimal Motion Planning“. In: *CoRR* abs/1105.1186 (2011). arXiv: 1105.1186. URL: http://arxiv.org/abs/1105.1186.

[24] Michael Seyfarth. *Lecture notes Control Engineering.* Institute for Control Engineering of Machine Tools and Manufacturing Units, University of Stuttgart, Germany. 2014.

[25] A. Loria, J. Dasdemir, and N. Jarquín-Alvarez. „Decentralized formation-tracking control of autonomous vehicles on straight paths“. In: *53rd IEEE Conference on Decision and Control.* 2014, pp. 5399–5404. DOI: 10.1109/CDC.2014.7040233.

[26] Hannes Wind. „Review of Formation Control Approaches and their Implementation Considering a Realistic Model of Mobile Robots“. Diploma Thesis, Institute of Systemdynamics, University of Stuttgart, Germany. 2016.

[27] Ming Lei, Shao-lei Zhou, Xiu-xia Yang, and Gao-yang Yin. „Complex Formation Control of Large-Scale Intelligent Autonomous Vehicles“. In: *Mathematical Problems in Engineering* 2012 (Dec. 2012). DOI: 10.1155/2012/241916.

[28] E. Panteley, A.A.J. Lefeber, A. Loria, and Henk Nijmeijer. „Exponential tracking control of a mobile car using a cascaded approach“. Undefined. In: *Proceedings of the IFAC Workshop on Motion Control*. Sept. 1998, pp. 221–226.

[29] A. Loria, J. Dasdemir, and N. Alvarez Jarquin. „Leader–Follower Formation and Tracking Control of Mobile Robots Along Straight Paths“. In: *IEEE Transactions on Control Systems Technology* 24.2 (2016), pp. 727–732. DOI: 10.1109/TCST.2015.2437328.

[30] Vojtech Vonásek. „A guided approach to sampling-based motion planning“. In: 2015.

[31] Thomas Bräunl. *Lecture notes Robotics*. School of Electrical, Electronic and Computer Engineering, University of Western Australia, Australia. 2019.

[32] James Ng. „An Analysis of Mobile Robot Navigation Algorithms in Unknown Environments“. Dissertation, School of Electrical, Electronic and Computer Engineering, University of Western Australia, Australia. 2010.

[33] I. Kamon and E. Rivlin. „Sensory-based motion planning with global proofs“. In: *IEEE Transactions on Robotics and Automation* 13.6 (1997), pp. 814–822. DOI: 10.1109/70.650160.

[34] Sisdarmanto Adinandra, Erik Schreurs, and Henk Nijmeijer. „A Practical Model Predictive Control for A Group of Unicycle Mobile Robots“. In: *IFAC Proceedings Volumes* 45.17 (2012). 4th IFAC Conference on Nonlinear Model Predictive Control, pp. 472 – 477. ISSN: 1474-6670. DOI: https://doi.org/10.3182/20120823-5-NL-3013.00060. URL: http://www.sciencedirect.com/science/article/pii/S1474667016314872.

Erklärung des Autors

der Masterarbeit mit dem Titel

Implementation of Formation Control with Optimal Path Planning and Collision Avoidance

Hiermit versichere ich,

1. dass ich meine Arbeit bzw. bei einer Gruppenarbeit den entsprechend gekennzeichneten Anteil der Arbeit selbständig verfasst habe,
2. dass ich keine anderen als die angegebenen Quellen benutzt und alle wörtlich oder sinngemäß aus anderen Werken übernommenen Aussagen als solche gekennzeichnet habe,
3. dass die eingereichte Arbeit weder vollständig noch in wesentlichen Teilen Gegenstand eines anderen Prüfungsverfahrens gewesen ist,
4. dass ich die Arbeit weder vollständig noch in Teilen bereits veröffentlicht habe und
5. dass das elektronische Exemplar mit den anderen Exemplaren übereinstimmt.

Stuttgart, den 23.10.2019

Marius Fink

Inofficial translation of the Declaration of Authorship („Erklärung des Autors“). Only the original German wording is legally recognized.

I hereby certify

1. that this thesis, resp. in a group project the appropriately labeled parts of this thesis, has/have been created by myself,
2. that no other sources than those referenced have been used, and that all ideas adopted from other work, literally or paraphrased, are appropriately labeled as such,
3. that this submitted thesis neither in full nor to a significant part has been part of any other examination process,
4. that I have neither published this thesis in full nor in parts, and
5. that the electronic version is identical to the hardcopy.