Comparisons of Platooning Control Methods without Inter-Vehicle Communication

Roshan Azam

Thesis submitted for the degree of
Master in Cybernetics and Autonomous Systems
60 credits

Department of technology systems
Faculty of mathematics and natural sciences

UNIVERSITY OF OSLO

Spring 2022

Comparisons of Platooning Control Methods without Inter-Vehicle Communication

Roshan Azam

Comparisons of Platooning Control Methods without Inter-Vehicle Communication

http://www.duo.uio.no/

Printed: Reprosentralen, University of Oslo

Abstract

Platooning is when one or more follower vehicles follow a lead vehicle autonomously based on information gained through sensors or communication. Due to being autonomous, the vehicles in the platoon can react faster and drive closer to each other than usual. The faster reaction increases safety. The smaller gap between the vehicles leads to less air drag, which reduces fuel consumption.

In this thesis, two methods for leader-follower platooning control without inter-vehicle communication were simulated and implemented on two Turtlebot3 Waffle-Pi robots. Both controllers only relied on visual information and were prescribed performance controllers. The first controller used the relative distance and bearing angle between the vehicles. The second controller used the pixel coordinates of a feature point in the follower's camera. Experiments were done where the robots drove in different patterns to evaluate the controllers.

The original controllers only used visual information. Modifications were made so the controllers could use a camera, LiDAR, or both. A moving average filter was added to reduce noise. The filter caused issues in some cases because it introduced a delay, which affected the system in situations where the velocity was dynamic. Using a LiDAR made the methods more robust toward problems that could occur from using the camera but worse at measuring angles. The LiDAR measurements were noisy, and the LiDAR in the physical system had a bias, but using it was more stable than using the camera in cases where the visual information was lost or affected by external factors.

Both controllers worked well in the simulations, with the pixel-based method being better. There were issues with the pixel-based method in the physical experiments due to it being sensitive to external factors that affected the camera, such as the robots driving over uneven flooring. Because of this, the distance-based method was more suited for physical implementation. Adding a LiDAR improved the physical pixel-based method's stability, as it negated the camera issues. If the trajectory of the robots in the experiments was curved, the pixel-based method would shortcut more than the distance-based method.

Contents

Acronym List

LIDAR Light Detection and Ranging

GPS Global Position System

RADAR Radio Detection and Ranging

SARTRE Safe Road Trains for the Environment

PATH Partners for Advanced Transportation Technology

GCDC Grand Cooperative Driving Challenge

V2V Vehicle to Vehicle

V2I Vehicle to Infrastructure

ITS Intelligent Transport System

ELROB European Land-Robot Trial

ROS Robot Operating System

MPC Model Predictive Controller

PI Proportional-Integral

PID Proportional-Integral-Derivative

FIFO First In, First Out

CTH Constant Time Heading

PPC Prescribed Performance Controller

ArUco Augmented Reality University of Cordoba

TF Transformation

FOV Field of View

RMS Root Mean Square

CSV Comma-Separated Values

FPS Frames Per Seconds

AoV Angle of View

Chapter 1

Introduction

Platooning is the linking of two or more vehicles in a convoy. The lead vehicle can be autonomous or human-driven. The followers are autonomously following the leader based on sensor data or communication. The platooning vehicles are essentially linked with invisible "chains" so they maintain a close distance to each other while in a platoon (ACEA, n.d.-b). This distance is often constant. Platooning does not always require the vehicles to follow directly behind the lead vehicle. They can also drive next to each other or in different formations and patterns. A form of platooning can be seen in nature. Some birds fly in a V formation because it conserves energy. They get less wind resistance, so they get less tired and can fly longer distances without rest. Flying in a V formation also lets the birds keep track of every individual in the formation, making it easier to communicate and coordinate (Science Reference Section, Library of Congress, 2019). Platooning in vehicles can be viewed as an advanced form of cooperative adaptive cruise control.

Platooning can be done longitudinally or laterally. Longitudinal platooning is when the vehicles follow directly behind the leader. In some platoons, each of the vehicles follows the vehicle in front of itself instead of a common leader. Longitudinal platooning is the most common. Lateral platooning is when the following vehicles are on the sides of the leader vehicle instead of directly behind. Different formations can be made by combining the two types of platooning.

Platooning can have many benefits, such as reduced air drag, lower fuel consumption, more security, less space between the vehicles, and increased driver comfort. Some of these benefits become more apparent when looking at cargo trucks. The reduced air drag from platooning leads to lower fuel consumption in trucks because air drag accounts for 25% of the truck's fuel consumption. Transport of cargo accounts for 1/4th of the greenhouse gas emissions in European cities (Turck, n.d.). In the U.S, trucks are responsible for almost 3/4th of the total freight energy use and greenhouse gas emission. It was discovered that 65% of the miles that these trucks drive could be platooned, leading to a potential 4% reduction

in total truck fuel consumption (Muratori et al., 2017). Truck platooning could potentially reduce CO2 emissions by 10% according to (ACEA, n.d.-a).

Platooning leads to increased safety. Human error is reduced when the vehicles become more automated. Almost 90% of all traffic accidents are due to human error (ACEA, n.d.-a). Platoons can react quickly to potential obstacles. The follower vehicles only need 1/5th of the time a human would need to react to the same situation, making platooning safer than if a human was driving.

Due to the faster reaction time, trucks in platoons can drive closer to each other and be as close as 12 meters. Therefore, platoons can use the roads more efficiently, as they take up less space than non-platooning vehicles. Using the road more efficiently leads to fewer traffic jams, and goods can be delivered faster (Pnorental, n.d.). A study from Korea in 2019 showed that truck platooning could result in annual benefits from travel time savings that corresponded to 167.7 million U.S $ in 2020. This study was done in a simulation based on Korean freeways with 3 or more lanes. (Jo, Kim, Oh, Kim, & Lee, 2019).

Platooning increases driver comfort, as the driver does not have to pay as much attention to driving while in a platoon (Turck, n.d.). Under current legislation, truck drivers are either driving or resting. If the vehicle is partially automated or self-driving, the driver can focus on other tasks, such as administrative work or making calls (ACEA, n.d.-a). As technology becomes more advanced, the follower vehicles could be fully automated, meaning they would not need drivers. According to (News, n.d.), there is currently a shortage of truck drivers, with a shortfall of 76,000 drivers in the U.K and 400,000 in total in Europe. Platoons could help alleviate this problem, as fewer drivers would be needed if the following trucks in the platoons were fully automated.

In truck platooning, it is common to assume that all trucks started in the same location and are heading to the same destination. It is also common to assume that the entire platoon is from the same business. The single-truck businesses can also benefit from platooning by making plans and joining platoons that are heading in the same direction (Fullbay, n.d.). The Global Truck Platooning Systems market size was at 1818.3 million U.S $ in 2020 and is projected to reach 9269.9 million U.S $ by 2027. Some of the key companies in platooning are Volvo, Peleton Technology, Scania, and Uber (WBOC, n.d.).

Platooning has many benefits, but there are also concerns and issues. Daimler, a large truck manufacturer that was heavily involved in platooning research, published a press release announcing that they were abandoning platooning and instead focusing on highly automated trucks (Saracco, 2019). The reason was that newer truck models are more aerodynamic. This made platooning less appealing, as one of the main reasons for platooning was the reduced air drag, which reduced fuel consumption. Another reason Daimler abandoned platooning was that it was impossible to keep a platoon together for the entire duration of travel. The splitting and re-merging of platoons meant that vehicles had to accelerate, which decreased

how much fuel was saved.

Reactions to platooning on the motorway were mixed, especially considering safety and the dangers that may come with platooning. (Road Safety GB, 2017) shows stakeholder reactions to platooning. While it mostly shows positive reactions from stakeholders, they also come with some concerns. These concerns were that platoons might block the view of important information, like signs, which disturb drivers outside of the platoon. A rush towards platooning might jeopardize the safety of motorways if not implemented correctly. The increase in cyber-crimes could lead to problems when vehicles are automated and rely on communication and other data. There were also concerns about platoons exiting and entering highways in busy areas with many exits/entries, and how regular non-platoon drivers should behave around platoons.

There are many ways of implementing platoons. It is often implemented in two parts; a control part and a vehicle tracking part. The follower tracks the leader's position through communication between the vehicles, with sensor data, or both. Some common tools used in platooning are cameras, Light Detection and Ranging (LIDAR), Global Position System (GPS), and neural networks.

In this thesis, there is one leader and one follower vehicle. The vehicles do not communicate, and the follower relies on sensors to track the leader. The vehicles are TurtleBot3 Waffle Pi robots and come with a Raspberry Pi camera and a 360-degree LiDAR. The camera is attached to the front of the robot. Turtlebot is a low-cost, beginner-friendly robot. The robots are well supported, straightforward to use, and have many ready-made packages and resources. The two Turtlebot3 Waffle Pis are used to implement platooning control methods. The methods only rely on visual information. They were first simulated and then implemented on a physical setup.

The most popular vehicle tracking techniques used to be LiDAR-only techniques, according to (Manz, Luettel, von Hundelshausen, & Wuensche, 2011). In the Defense Advanced Research Projects Agency (DARPA) Urban Challenge 2007, mainly LiDARs were used for vehicle tracking. Using only a LiDAR can be challenging when there's more than one object in front of the follower, as the LiDAR struggles to tell the difference between the lead vehicle and other objects. Because of this, it can be better to use a camera instead. The problem with using a camera is that if the lead vehicle leaves the Field of View (FoV) of the camera, or the light and weather conditions are bad, vehicle tracking can be difficult. It can be hard to track distances with only a camera. It is possible to combine the camera with a LiDAR to let the camera do the main tracking and let the LiDAR find the distance. Radio Detection and Ranging (RADAR) can also be used to find the inter-vehicle distance between the lead and follower vehicles, as shown in (Kim, Jang, Jang, & Kim, 2020), where they present a camera and RADAR-based perception system for platooning. They combine the RADAR with a camera because RADARs have a hard time tracking laterally, and it is difficult to know what type of object is detected using only a RADAR.

In this thesis, first, earlier works concerning platooning will be presented. Then a closer look at some ways of implementing platooning, both with and without inter-vehicle communication. After that, theory and a closer look at two methods of platooning without communication will be presented. Additions to the methods, like a filter and a LiDAR, will also be made. These methods will then be simulated and implemented, and experiments will be done. Finally, the results of the experiments will be presented and discussed. The goal of this thesis is to simulate and implement the controllers, find the characteristics of the controllers, and compare them.

Chapter 2

Earlier works

When doing platooning, there are two main applications. The first is platooning on highways. This field has been well researched. The second is off-road platooning. This will be the focus of this thesis. This field is not as well-researched, and the biggest difference between the implementations of the two applications is usually what type of sensors and equipment is used. Platooning on highways often uses communication in addition to cameras and other sensors, like LiDAR or RADAR. Here, every vehicle in the platoon can communicate with each other. In off-road platooning, there is often no communication. Instead, the following vehicles track and follow the leader based solely on information obtained by their sensors. This type of follower vehicle is often called the "ego"-vehicle. In this chapter, some of the earlier work that has been done will be presented.

2.1 Earlier projects and challenges

Many researchers have been looking into platooning on highways with trucks and off-road platooning. Several big projects have been done on platooning and the tracking of vehicles. Challenges have also been held to test platoons and see how far the research field has come.

Safe Road Trains for the Environment (SARTRE) was a project funded by the European Commission. It explored platoons operating on highways, where they had mixed platoons consisting of both cars and trucks. SARTRE's main goal was to have platoons where the drivers could do other tasks instead of driving. While convenience was the main goal of SARTRE, it also had safety, reduction of fuel consumption, and reduction of energy as goals. Another important goal for SARTRE was to avoid changes to infrastructure so motorways did not have to be re-built with special lanes or have special equipment to accommodate the platoons. Vehicle to vehicle (V2V) communication was a vital part of the platooning

system for SARTRE, and SARTRE dealt with both longitudinal and lateral platoons. This was a three-year project that started in 2009 and ended in 2012. (Bergenhem, Shladover, Coelingh, Englund, & Tsugawa, 2012).

California Partners for Advanced Transportation Technology (PATH) is a research and development program founded in 1986 at the University of California. PATH's main goal was to increase the throughput of vehicles so highways did not have to be reworked to fit the growing number of vehicles on the road. Platoons would take up less space because the vehicles could be closer to each other. The platoons consisted of either cars or trucks but did not mix the two in one platoon. The platoons would also have dedicated lanes on the highway. PATH's studies showed that with platooning, they could fit 2-3 times more vehicles on the highway if platoons of up to ten cars were made. (Bergenhem et al., 2012) (Gallagher, n.d.) (UC Berkeley, n.d.).

In 2011, the Grand Cooperative Driving Challenge (GCDC) was held. The goal was to accelerate the development and integration of cooperative driving systems based on a combination of V2V and vehicle to infrastructure (V2I) communication. The challenge was to increase the throughput of vehicles by reducing the spacing between vehicles. And to demonstrate how traffic "shock waves" can be weakened. These shock waves propagate along a line of vehicles and are generated by collisions or changes in velocity. They are a result of traffic conditions changing (University of Idaho, n.d.). GCDC was held again in 2016; the challenges were cooperative platoon merging and cooperative intersection passing. (Bergenhem et al., 2012) (Englund et al., 2016).

Energy-Intelligent Transport System (Energy-ITS) was a project by the Japanese Ministry of Economics, Trade, and Industry. The goal was to save energy. The slower a vehicle is traveling (up to 40-50 km/h), the more CO2 it emits. With ITS, vehicles can move at more efficient speeds, making it so they emit less CO2. Another goal of Energy-ITS was to mitigate the lack of skilled drivers on the road by having ITS help the flow of traffic. The project started in 2008 and ended in 2012. (Bergenhem et al., 2012) (ITS Asia-Pacific, n.d.).

In 2016, the European truck challenge was held for the first time by the Netherlands to promote platooning by bringing truck convoys to public roads for the first time. Six truck manufacturers would bring platoons of semi-automated trucks, crossing borders from different European cities to reach their destination of Port of Rotterdam (Acea, 2016). All six platoons succeeded, and the challenge cleared the way for real-life convoys. The European truck challenge was not held to test the technological aspect of platooning, as the organizers were certain the technology was already there. It was held to prove that they could overcome the institutional and operational challenges of running platoons in different countries. (Roberts, 2016).

One of the earliest projects involving platooning was the German national project, KONVOI. The project lasted from 2005 to 2009 and focused on the platooning of heavy trucks

with small gaps between each truck to improve aerodynamics. KONVOI studied driver acceptance, traffic flow, environment, and legal and economic implications of platoons. The platoons were successfully developed and tested in 2009 and drove 3100 km in real traffic, making it the first platoon system to be tested in real traffic. The KONVOI project was inspired by the PROMOTE CHAUFFEUR I+II projects, which were the first projects in the EU that dealt with platooning. The PROMOTE CHAUFFEUR I+II projects focused on the technical feasibility of platoons. (TRIMIS, 2021).

European Land-Robot Trial (ELROB) is the longest-running field robotics and unmanned systems event. It has been held since 2006. ELROB has trials reflecting real-world scenarios. The event's main purpose is for companies to demonstrate their equipment and for users to see the equipment used in real-world scenarios (Schneider, Unknown). One of the ELROB trials is a convoy scenario, where a highly automated vehicle follows a leading vehicle through an unknown, unstructured domain. The leading vehicle is driven by a human through a non-urban area. The winner of ELROB 2016 was Team MuCAR (Munich Cognitive, Autonomous Robot Cars), their work will be further examined in section 2.3.1.

These were some projects and challenges involving platooning. There have been other companies that have researched platooning that have not been looked at thoroughly here, like Scania, Daimler Truck company, and DAF Trucks. (Dekra, 2018)

2.2 Platooning with communication

Even though this thesis will be focusing on platooning without communication, it is worth going over some important works and concepts on platooning with communication to understand platooning better. Communication in platooning mainly happens through V2V and V2I communication.

An important concept in platooning, especially on highways and with several vehicles, is string stability. When the lead vehicle in a platoon slows down, the following vehicles should also slow down in such a way that they avoid a collision. A platoon is string stable if disturbances are not amplified when they are propagated over vehicle strings. If the disturbances are amplified, the platoon is not string stable. String stability is often used to evaluate the performance of a platoon since it can show whether a platoon can handle changes in traffic conditions. When it comes to research on off-road platooning, there is little mention of string stability. It is hard to say why, but it might be because there is less emphasis on traffic in off-road scenarios. (Feng et al., 2019).

A problem when using communication in platooning is communication delays. The performance of the platoon is connected to the communication link. Because of noise and reflections, information can be lost. This can lead to collisions within the platoon and loss of

string stability. In (Vinel, Lyamin, & Isachenkov, 2018), they try to solve the issues caused by the delay by making a model for the maximum tolerable delay that a platoon can have before it starts having problems.

2.2.1 Implementations of platooning with communication

There are many ways of implementing platooning with communication. In (Rezgui, Gagné, Blain, St-Pierre, & Harvey, 2020), it is implemented by having a V2I communication algorithm and using a camera to determine the position of the following vehicle relative to the leader. Ultrasound is used to determine the distance between the vehicles, and a convolutional neural network is used to classify a pink marker behind the leader. They have a custom-built robot with an Arduino. Data is transferred from different sensors to the infrastructure of the robot. The infrastructure then returns the decision that the robot should make.

In (Torabi & Wahde, 2018), a method to optimize fuel efficiency is given. Instead of having a fixed desired distance between the vehicles in a platoon, the speed profile of every vehicle is calculated based on the route they are driving. The optimization is done through evolutionary algorithms that ensure a minimum distance is kept to avoid collisions.

Vehicles in a platoon have three main actions, follow, merge and split. (Farag, Mahfouz, Shehata, & Morgan, 2019) introduce a Robot Operating System (ROS) based protocol for vehicles merging with or splitting from a platoon and show an algorithm for these actions. (Bergenhem, 2015) compares three methods of sharing information within a platoon. The standard method uses Cooperative Awareness Messages (CAM) and Decentralized Environmental Notification Messages (DENM). The two new methods they propose are a minimal and a full protocol. The new protocols address issues with the standard method; they have a fixed update rate and extra data. The full protocol has two-way communication through handshakes. The minimal protocol and standard method only have one-way communication.

In (Zhao, Yao, Li, & Wang, 2017), a system for following a lead vehicle by tracking the leader's path is designed. This path is made up of waypoints that give information about the leader's position and velocity; these are collected in intervals. The system mimics humans by continually locating the leader and remembering the leader's path. The waypoints are managed by a waypoint management algorithm. The waypoints are pushed into a queue when the leader's pose is updated; it is updated at a fixed interval. The leader is equipped with a GPS, inertial navigation, and odometer. An integrated Kalman Filter fuses all the measurements for pose estimation and uses communication to transmit the pose. The vision-based tracking initializes a region of interest with the help of the GPS and then uses Histogram of Oriented Gradient (HOG) features. Combining this with LiDAR-based tracking, inter-distance control, and trajectory following with object avoidance gives robust convoys.

As previously mentioned, there can be communication problems such as delays. Another problem is that the IEEE 802.11p protocol that is used for vehicle communication can be vulnerable to attacks and jamming. In (Ucar, Ergen, & Ozkasap, 2018), they try to alleviate the communication issues by using Visible Light Hybrid Communication. This method is a hybrid between Visible Light Communication (VLC) and IEEE 802.11p. While this works, they state that relying only on VLC can lead to degradation in platoon stability because it is sensitive to environmental effects. In (Abualhoul, Marouf, Shagdar, & Nashashibi, 2013), they study the applicability of VLC, where they build a complete model and investigate if this is a good method for platooning. They use the rear lights of vehicles as a communication link. They managed to get a Bit-Error-Rate of 10^{-6}, but they had problems when the roads curved too much.

2.3 Platooning without communication

As mentioned, communication cannot always be relied upon. Therefore, other ways of making platooning systems that do not require communication must be explored. Since there is no communication in these methods, there will be more weight on sensors like the LiDAR and cameras. Platooning without communication can be implemented in different ways.

In (Benhimane, Malis, Rives, & Azinheira, 2005) they present a complete platooning system with the use of visual tracking in an outside environment. The visual tracking is done by estimating the homography between a selected reference template attached behind the lead vehicle and the corresponding area in the current image. The relative pose is found by using homography decomposition. For the control part, they use kinematic modeling with Cartesian coordinates and path tracking. They also have an alternative for more robust distance-based lateral path tracking. The control objective is to follow the leader's path, and the path tracking error is given in Cartesian coordinates.

In (Yang, Liu, & Jiang, 2018) they show an autonomous multi-vehicle following queue control system that uses vision and lasers, simulated in ROS. Vision and laser measurements are fused to keep the followers at a desired distance from the lead vehicle. The control is done by finding control laws from kinematic models. The model is based on a 2-dimensional top-down coordinate system and looks at the error between the follower's desired position behind the lead vehicle and its actual position. Two parameters must be found for the control law; relative orientation and distance. The relative orientation is found with the camera by finding the centroid of the target and looking at the orientation between the target and camera source. The Camshift algorithm is used to track the target and decide if it should be followed or not and can be used when the centroid is found. The relative distance is found using the laser.

In (Baardseth, n.d.) they presented methods for vehicle detection and pose estimation.

They find the pose of the lead vehicle relative to the ego-vehicle by using Pose from Orthography and Scaling (POS) and POS with iterations (POSIT). These methods require a minimum of 4 known 3D feature point pairs and their corresponding 2D image coordinates. Image geometry is used to convert 3D to 2D points, and Random Sampling Consensus (RANSAC) is used to find good data points/inliers. Based on the visual and geometric properties of the vehicles, 3D point cloud detection can be done with LiDAR. Using the point cloud and the properties of the vehicles, relative pose estimation can be done. They have two alternative methods for finding the relative distance between vehicles. One method uses LiDAR, and the other uses POSIT. For model-based pose estimation, they use either a model of the license plate or the vehicle.

A method that uses pre-trained neural networks is presented in (Mutz et al., 2017). The method tracks the lead vehicle by using a Deep Neural Network (DNN) to find a bounding box in the current image that is most likely to contain the lead vehicle. Then a LiDAR is used to find the depth information since LiDARs are good at finding distances. Once the current position of the leader is estimated, it is used to track the leader's path. The position estimate is evaluated, and if it is an inlier, it is incorporated into the leader's path. The method uses a localizer module to compute the follower's current position in the environment. Localization is done by using a GPS-Real Time Kinematic (GPS-RTK), an odometer, and a LiDAR. Then a Monte Carlo Particle Filter is used to compute the follower's pose. The method has a mapper module to create representations of which areas do not have obstacles, a module path tracker to track the leader's path using the LiDAR, and a neural network. It also has modules for obstacle avoidance, behavior selection, motion planning, and a low-level controller. A big advantage of the method in this paper is that it does not require models or adaptations of the leader vehicle.

2.3.1 ELROB 2016

In this section, the winners of the ELROB 2016 trial will be discussed in more detail. Team MuCAR has been researching autonomous vehicles for more than 30 years and has participated in earlier ELROB trials with great success. (Elrob, n.d.).

In 2011, they released a paper (Manz et al., 2011) on how to do monocular model-based 3D tracking that can deal with complex lighting, partial occlusion, and cluttered color images. It is a 6DOF vehicle tracking problem. Tracking with only a LiDAR can be unreliable. Because of this, they use a 3D model of the lead vehicle, which can result in a good tracking method using only a camera. To accurately track the leader, the algorithm is given the leader's shape and appearance in a 3D model, which consists of vertices, edges, and colored regions. They use a camera mounted on a moving platform and utilize a 4D approach in addition to a particle filter.

They build further upon their work in (Fries, Luettel, & Wuensche, 2013). They present a

template-based solution using different features to estimate 3D pose roughly but fast. They then combine this method with their previous work, making a fast and accurate tracking method. They do not use LiDAR or RADAR because it is expensive. They use their model-based tracking system to do the main tracking, but this method needs to be re-initialized at the start/at a loss of visual detection. For this purpose, they build a template-based solution. The template-based tracking uses 3 sides of the vehicle that should be tracked (two sides and the back). The template-based tracking estimates the pose and velocity based on this. It uses feature matching, tries to match the template to the images in the camera, and uses a classifier to improve the performance of the tracking. Then a Kalman Filter is used to predict the vehicle pose and region of interest to improve the results. They then combine the model- and template-based tracking methods.

In a later paper, they improve on their template-based model by adding pre-processing steps that remove unnecessary image information. The improvements are made by attaching 2D templates to 3D models, adding image features, and training cascade classifiers. This process generates a 3D template model. After the 3D template is generated, a Scaled Unscented Kalman Filter is applied to predict the relative vehicle pose. In addition, they use rotated image features to enhance pose estimation. (Fries & Wuensche, 2014).

They extend their tracking system by using more than one camera sensor in (Fries & Wuensche, 2015). They use low-light, thermal and daylight cameras to be able to detect vehicles at nighttime. In addition, they get higher stability and accuracy by coupling a Kalman filter with their particle filter. Apart from that, the tracking method is very similar to their previous methods. They add a LiDAR to get depth information. The LiDAR is also used if the lead vehicle drives out of the camera's FoV.

In the ELROB 2016 trial, they mainly made use of the methods mentioned in this section and fused measurements from different sensors. Their system had mission planning, vehicle tracking, and path generation. The mission planning contained the convoy leader type, the follower's maximum velocity, and the minimum relative distance. For tracking, they had tracking with a LiDAR-only method and a method for camera and LiDAR-based tracking. They combined these with data fusion at the object level, which was done by a Gaussian Mixture Probability Hypothesis Density filter. The path generation generated a path for the follower based on ego-motion information and the tracking of the leader. (Fries et al., 2017).

2.4 Control

Until now, mainly the vehicle tracking aspect of platooning has been discussed. Another important part of platooning is the control aspect. Platooning control is about how autonomous vehicles are controlled based on sensor input and other information. The focus of this thesis will be on control methods that do not require any communication. Many

different strategies for control exist; some of them will be mentioned in this section. In this thesis, two methods for platooning control without inter-vehicle communication will be chosen, discussed in greater detail, simulated and implemented, and compared.

In (M'Sirdi, 2021), they show an overview of different control models. The main difference between the models is what assumptions are made and what information is neglected. 1D models are mostly used to study inter-vehicle safety distance, where it is assumed that the platoon is in a straight line with no lateral movement, only longitudinal. Some 1D models are the longitudinal double integrator, the longitudinal unidirectional model with simple mass, and the longitudinal bidirectional model with a mass-spring-damper.

2D convoy models are more useful as they model both longitudinal and lateral movement. The kinematic model is a popular 2D convoy model; here the dynamics are neglected. The unicycle model, which uses a kinematic representation, is simple and used for vehicles with two wheels on an axle. The bicycle model, or Ackermann's model, is used for vehicles with two axles, where the front wheels are controlling the lateral movement of the vehicle. There is also the robotics model, which is composed of dynamic equations, kinematic transforms, and geometric representation, which makes it more precise. In (M'Sirdi, 2021) they suggest a new control model, saying that the current models that exist are too simple. They point to how certain animals swarm or move in packs and that the formations are based on the species and other factors. They propose a method that takes the geometrical differences of the platoon vehicles into account.

In (M'Sirdi, 2021), they also show an overview of different strategies that can be used for control, saying that these strategies can be classified into kinematic/dynamic, local/global, uni, or bi-directional. The classifications are based on what information is used to control each vehicle. For example, global methods use sensor information from all vehicles, while local methods use information from neighbors only. Sometimes if the control laws are robust, precise models are not needed (Nacer, Dahmani, & Nasser, 2020). Another classification of control systems is decentralized/centralized. Decentralized control systems have no communication, while centralized control systems have communication (Yazbeck, Scheuer, Simonin, & Charpillet, 2011).

Previously, how (Benhimane et al., 2005) implemented their vision-based control for car platooning by using homography decomposition was presented. As mentioned, they use a kinematic model with Cartesian coordinates. For control, they consider a car-like vehicle that purely rolls without slipping, so the velocity vector always points perpendicular to the rear wheel axis. They describe this kinematic model with the help of absolute Cartesian coordinates, the longitudinal velocity, heading angle, and steering angle. They do path tracking by attaching a moving reference frame to the lead vehicle and having a current frame for the follower. The origin of the current frame is the current position of the follower, and the origin of the reference frame is the desired position. The difference between the origins is the path tracking error. By computing the derivatives of the error and combining

this with the kinematic model, they get a local kinematic model. By using a simplified 1st order model for vehicle dynamics, they get a model that is decoupled into a longitudinal and lateral model. They swap the lateral model for a distance-based lateral model for more robustness. To adapt the control approach to platooning, they set a reference frame rigidly linked to the leading vehicle as the control objective.

Previously, (Yang et al., 2018), and their implementation was mentioned. The control in their paper was done by finding control laws for the kinematic model and looking at the error between the desired position of the follower behind the lead vehicle and the actual position. They use a Proportional-Integral-Derivative (PID) controller to control the velocity of the wheels of their robots. In (Mutz et al., 2017), as previously mentioned, they used different modules. The most important module for control is the low-level controller module. This module maintains two independent controllers, one for velocity and one for steering. The velocity controller is a standard PID controller, and the steering is done by a Model Predictive Controller (MPC). The MPC simulates the output some time steps ahead to decide which steering wheel effort to apply in the current time step. They used a Neural-Based MPC because the standard MPC could not handle latency and non-linearity. Another important module for control is the motion planner. It receives a goal list produced by a behavior selector module and computes a set of motion commands to achieve the goals. Each motion command consists of a linear velocity, steering wheel angle, and duration for how long the command should be applied.

In (Scheuer, Simonin, & Charpillet, 2009), a longitudinal controller is presented. This controller is then built upon by combining it with lateral control in (Yazbeck et al., 2011). Lateral and longitudinal control can be dealt with separately. In (Scheuer et al., 2009), they started by taking the controller presented in (Daviet & Parent, 1996) and improving upon it. The controller in (Daviet & Parent, 1996) computes acceleration as a function of the follower's velocity, the follower's distance to the lead vehicle, and the lead vehicle's velocity. In (Scheuer et al., 2009), they improved this controller by ensuring collisions were avoided by setting some constraints. In approaches to platoon control without inter-vehicle communication, there can be issues with lateral deviation, which is accumulated along the platoon. In (Yazbeck et al., 2011), they improve on this and minimize lateral deviation. They use a kinematic model and assume that the wheels of their robots do not slip and use odometry and a sensor like a camera/laser. The lateral control consists of the follower memorizing the leader's path. The leader's position in a fixed frame corresponding to its configuration at the start of the experiment is put into a First In, First Out (FIFO) list. The positions in the list give a linear approximation of the path that the follower tries to follow. Curvilinear distance to this path is computed and updated, and the FIFO list is updated as the follower follows the path. Control law computation is used to track the leader's path more robustly by using it to reach a target position in the memorized path between the two robots. The target is the first position whose inter-distance (distance between the vehicles) with the current position is higher than a look-ahead distance. Since the position of the lead vehicle is given in a fixed frame and computed according to the odometry, it will suffer

from drift, and the position will become less accurate. The memorized position is not used directly, only its backward projection. Therefore, the drift can be neglected.

In (Ali, Garcia, & Martinet, 2015) they use a Constant Time Heading (CTH) policy for longitudinal control and sliding mode control for lateral control. The CTH policy is given in (Swaroop & Rajagopal, 2001) and is a variable spacing policy where the desired following distance is proportional to the speed of the vehicle. In (Ali et al., 2015), they make a longitudinal dynamic model by using the control law given in (Sheikholeslam & Desoer, 1993) and improve it by taking the model of the engine into account. For the lateral model, they use the bicycle model and reformulate it by adjusting the model so that the steering is not instantaneous. From this, they get a linear model for the lateral system. Both these controllers need control inputs, so one input for the longitudinal controller and one for the lateral controller. They propose a new way of modeling platoons with a flatbed tow truck model. Here the forces between each vehicle in the platoon are represented by a one-directional spring-damper system for the longitudinal controller. They add a virtual truck into the model. The truck is assumed to move at a certain velocity, and the platooning vehicles' velocity must be subtracted from the truck's velocity. They can then deal with relative velocity instead of absolute velocity. The lateral platoon model consists of two springs and two dampers. The springs and dampers model the error in the angle/heading and the distance between the lead and following vehicle. By using a modified CTH and a curvilinear spacing error, they find the control input for the longitudinal controller. By using the sliding mode control, they find the control input for the lateral controller.

In (Kwon & Chwa, 2014), they use coupled sliding mode control, which is presented in (Park & Chwa, 2009). In (Kwon & Chwa, 2014), they describe an interconnected system with a constant distance between the vehicles, which is described by a non-linear dynamical model given in (Stankovic, Stanojevic, & Siljak, 2000). From the dynamical model, (Kwon & Chwa, 2014) design bi-directional control laws so that each vehicle keeps the desired distance to the preceding vehicle and so string stability is guaranteed. They propose an adaptive bi-directional control law using coupled sliding mode control, where the control objective is to make the distance error converge towards zero. They first choose sliding surfaces, which are needed to use the coupled sliding mode control. They can find information about the derivatives of the sliding surfaces by filtering. The convergence of the sliding surface of one vehicle to zero does not guarantee string stability, so they choose the coupled sliding surface of the vehicles for control of the entire platoon. Using information about the sliding surfaces, they design a control law that makes the coupled sliding surface converge to zero. They also use parameter adaption laws for unknown variables in the control law. When the control law is employed in the interconnected system, the distance errors converge to zero and guarantee string stability if a weighting factor in the coupled sliding surface is between 0 and 1.

In (Petrov, 2008), they propose a mathematical model for control of an autonomous vehicle convoy. They use the bicycle model and assume that the wheels of the vehicles do not slip. They put non-holonomic constraints on the model. Through this, they create a kinematic

model of the lead and following vehicles after doing some transformations and setting up coordinate systems. A reference point is set between the vehicles. The point is some distance behind the leader and some distance in front of the follower vehicle. Taking the difference between the kinematic models of the leader and follower gives a relative kinematic model with the error in position and angle. They take the derivative of the model to get the error of the system in the next time step, giving them the inter-vehicle kinematics in error coordinates. The velocity of the lead vehicle is unknown and unavailable for feedback control design. Because of this, they propose an adaptive control law that achieves stabilization of the system in error coordinates. To do this, they start by defining the kinematic models and reducing them to adaptive control design. They use a Lyapunov function candidate and derivate it. They then choose variables for the linear and angular velocity of the leader so that every term containing them is eliminated from the function. This gives an adaptive control system.

In (Belkhouche & Belkhouche, 2005), they use guidance law strategies to make control laws for platooning. The guidance laws are velocity pursuit, deviated pursuit, and proportional navigation. These guidance laws are used to derive control laws for the velocity and angular velocity of the follower. They start by creating a kinematic model for the robots in Cartesian coordinates and transferring it to polar coordinates. They look at the relative velocity between two robots and use this to derive a kinematic model for the platoon. They then use the guidance laws, which are based on geometry and kinematic equations. The velocity pursuit is about making the velocity vector of the pursuer lie on the line of sight joining the pursuer and target. Deviated pursuit is when there is a non-zero angle between the line of sight and the velocity vector. Proportional navigation is a generalization of the velocity pursuit, where the angular velocity of the pursuer is proportional to the rate of turn in the sight angle. Using the laws, they get a control law based on the velocity pursuit because it can be shown that the two other laws can be generalized to become the velocity pursuit. They then create a second control law, which is derived from considering the kinematic equation between successive robots under the guidance laws. The second control law is for maintaining a constant distance between the vehicles.

Using the control method based on velocity pursuit in (Belkhouche & Belkhouche, 2005), (Hung, Vinh, & Dung, 2012) creates a simple platoon control system. The system uses a Sensory Input Map, where the follower can use a laser to find the relative distance and angle between itself and the lead vehicle. Using the relative angle and distance between the two vehicles, they can find the velocity and angular velocity of the lead vehicle.

In (Verginis, Bechlioulis, Dimarogonas, & Kyriakopoulos, 2015), they make decentralized control laws where no communication is required, using only visual feedback. They assume that the visual feedback is limited, meaning the follower can only detect the leader in a limited cone of vision. This method uses a unicycle model and only requires the distance and bearing angle between the leader and follower to maintain a platoon. They add constraints to the distance and bearing that help keep the platoon connected even with limited visual

feedback. Based on the relative distance and bearing, they define distance and heading errors. The controller used here is a prescribed performance controller (PPC). The prescribed performance characterizes the evolution of the errors and is defined by a steady-state performance and some positive parameters. The positive parameters are based on the min/-max distance and bearing between the leader and follower. They then use the steady-state performance function to normalize the errors and use these in the control laws.

In (Miao et al., 2021), they present a vision-based formation control method with FoV constraints. This method only requires visual feedback and uses a unicycle model. The method is based on detecting a feature point on the leader and making the pixel coordinates of the feature point coincide with some pre-determined, desired pixel coordinates. The desired pixel coordinates are chosen according to the desired position of the follower relative to the leader. They use a PPC similarly to (Verginis et al., 2015), but they deal with pixel coordinates instead of angles and distances. In (Miao et al., 2021), they present two controllers. A Nussbaum Gain Adaptive controller and a Static Nonlinear Gain controller and compare them. The Nussbaum Gain Controller has a dynamic control law, has control matrices instead of constants, and is a little more advanced. In their results, when comparing the two controllers, they found that the Static Nonlinear Gain controller gave the best results.

Chapter 3

Background

This chapter will go deeper into two chosen control methods and look closer at how they work. This chapter will also go over the theoretical background and the tools that are important to implement and understand the methods

The first method is the one shown in (Verginis et al., 2015). The method does not use communication and only relies on the relative distance and bearing angle between the two robots. The second method is shown in (Miao et al., 2021). Here, the relative distance and bearing are not needed, but the pixel coordinates of a feature point on the leader in the follower's camera are required. Both methods use the unicycle model and only require visual feedback. The unicycle model is required when working with a Turtlebot3 because the robot is a differential drive robot. A differential drive robot has two drive wheels mounted on a common axis, and each wheel can be driven individually; forward or backward (Dudek & Jenkin, 2010). The methods were first simulated in Gazebo and later implemented on the physical Turtlebot3 robots.

These methods were chosen because they only use sensors that are already available on the Turtlebot3, meaning a camera or LiDAR. Camera-based controllers were chosen because most recent methods mainly use a camera, with a LiDAR sometimes used as a supportive sensor to find distances. Many other methods rely on additional information, such as communication or a way to find the leader's heading. The chosen methods only rely on detecting the leader and finding the relative distance and bearing angle between the robots or minimizing pixel positions. Another reason for picking these methods is that they use the unicycle model to derive the control equations, as the Turtlebot3 must be modeled after the unicycle model. The methods also had to be recent. The two methods that were chosen are from 2021 and 2015. In addition, the controllers could not be too simple. A normal PID controller would, for example, be too simple. The two methods chosen here were good candidates.

3.1 Theory and tools

Before going deeper into how the controllers work and how they are implemented, background theory and the tools that are being used will be presented in this section.

3.1.1 Controllers

The controllers that are being implemented in this thesis both fall under the classification of prescribed performance controllers. PPC was first proposed in (C. Bechlioulis & Rovithakis, 2008) as a tool to priori characterize the transient and steady-state performance. Usually, repeated parameter regulation is required to get a good transient and steady-state performance. In the PPC, some transient and steady-state performance indicators like the overshoot, undershoot, and convergence rate of the system are characterized a priori by a performance function designed by the user. This makes it so that complex parameter regulation is not needed, lowering the complexity of the control design. The PPC method generates a controller that guarantees the preassigned performance set by the user. The performance function is designed offline. (Wei, Chen, Liu, & Yin, 2020).

In (Verginis et al., 2015), they use concepts and techniques from (C. P. Bechlioulis & Rovithakis, 2014). In (Miao et al., 2021), they use concepts from (C. P. Bechlioulis, Heshmati-alamdari, Karras, & Kyriakopoulos, 2019). In (C. P. Bechlioulis & Rovithakis, 2014), they design a universal, approximation-free state feedback control scheme that guarantees output tracking with prescribed performance and bounded closed-loop signals for any initial condition. This control design has low complexity and does not use prior knowledge of system nonlinearities or upper/lower bounding functions. An Image-Based Visual Servoing scheme is proposed in (C. P. Bechlioulis et al., 2019), which can guarantee prescribed transient and steady-state performance while satisfying FoV constraints despite camera calibration and depth measurement errors.

According to (C. P. Bechlioulis & Rovithakis, 2014), prescribed performance means that the output converges to a predefined arbitrarily small residual set with a convergence rate less or equal to a predefined value and with a maximum overshoot of a preassigned level. These concepts are then adapted to achieve a predefined transient and steady-state response for the errors while also avoiding violation of collision and connectivity constraints (Verginis et al., 2015).

In (Miao et al., 2021) two controllers are proposed and tested. One is a static nonlinear gain controller. The other is an adaptive controller based on Nussbaum gain. The Nussbaum gain technique is a tool for handling unknown control directions or coefficients, and the Nussbaum gain controller is a dynamic control law. The Nussbaum gain can sometimes be very large, and in the Nussbaum gain controller, two 2x2 matrices have to be tuned instead of the two

constants that have to be tuned for the static nonlinear gain controller. The Nussbaum gain controller also needs to calculate the inverse of a matrix, while the static nonlinear gain controller obtains all necessary information from the image errors. Because of this, the static nonlinear gain controller is more attractive according to (Miao et al., 2021).

The static nonlinear controller can handle unknown feature height, and it is static in that it is free of any parameter adaptation. It is based on a lemma in the presentation of a nonlinear Proportional-Integral (PI) controller shown in (Ortega, Astolfi, & Barabanov, 2002). The lemma says that there is a class of functions that can be used instead of the original function in one of the terms of the PI functions. The nonlinear PI controller is based on having a perturbation function that can be driven to zero. When choosing the functions that define the PI controller, they "shape" the perturbation function to exhibit at least one root, and trajectories are forced to converge towards this root.

In (Miao et al., 2021), both controllers were chosen because the feature height was unknown. Only the nonlinear static gain controller is implemented in this thesis because it gave the best results in (Miao et al., 2021). It had a smaller overshoot, a faster convergence speed of the formation error, smaller control inputs, and better performance.

3.1.2 Image geometry and pinhole camera model

To implement the controller shown in (Miao et al., 2021), knowledge about imaging geometry and projection is necessary. If there is a point in the real world seen through a camera, what would its pixel coordinates in the image be? And if a point is given in the image instead, how could the real-world coordinates of the point be found based on pixel coordinates? To model this, the pinhole camera model is used. It is the simplest camera model and describes the projection of points in a 3D space onto an image plane.

To transform the world coordinates of a point to pixel coordinates, the transformation must first be done from world coordinates to camera coordinates. Then from camera coordinates to image coordinates, and finally from image coordinates to pixel coordinates. To transform from pixel coordinates to world coordinates, the transforms would have to be done in reverse order. The image and pixel coordinates are 2D coordinates, while the world and camera coordinates are 3D coordinates. Figure 3.1 shows the pinhole camera model and the different coordinate frames.

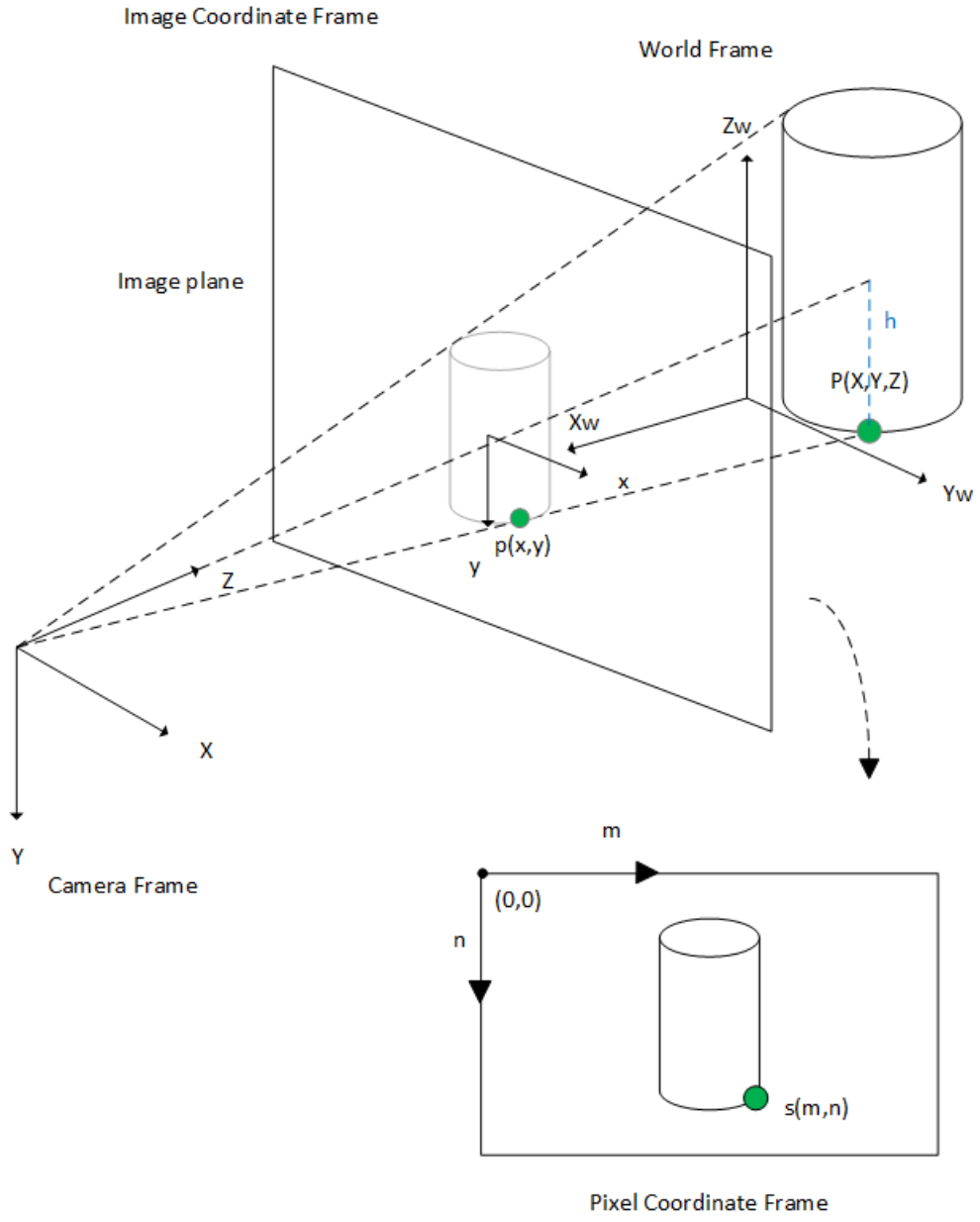

Figure 3.1: A figure of the pinhole camera model. It shows the different coordinate frames in the camera model and how a point on an object is projected into the image plane and then to the pixel coordinate frame. The green point is a feature point. h is the height difference between the point and the optical axis of the camera. Image inspired by Figure.1 in (Fernandez et al., 2017).

Assume there is a point in the world frame given by $\boldsymbol{P} = (X, Y, Z)$, where X, Y, and Z are known. Under the pinhole camera model, this can be projected into the image coordinates $p = (x, y)$ by $x = f * \frac{X}{Z}$, $y = f * \frac{Y}{Z}$. Here, f is the focal length (Collins, 2007). Z is called the depth. In the case Z is unknown, stereo cameras or other methods can be used to find it.

3.1.3 ArUco

An Augmented Reality University of Cordoba (ArUco) marker is a square marker composed of a black border and a white inner binary matrix that determines the marker's ID. The IDs are predefined, and there are dictionaries of different IDs for markers. The black border allows for fast detection of the marker in images, and the binary codification allows for the identification and application of error and correction techniques. The markers were developed in 2014 as a fiducial marker specially appropriated for camera pose estimation (Garrido-Jurado, Muñoz-Salinas, Madrid-Cuevas, & Marín-Jiménez, 2014). The benefit of using an ArUco marker is that a single marker and its four corners are enough to obtain the camera pose. In addition, the inner binary codification makes them robust. The dictionaries' main properties are the dictionary size, which is the number of different markers that compose the dictionary, and the marker size, which is the size and number of bits the marker has. For example, a 4x4 marker has 16 bits. (OpenCV, n.d.).

When it comes to marker detection, it is comprised of two steps. First, the detection has to check what is and is not a marker in the image and find potential marker candidates. This is done by checking for squares using computer vision techniques like adaptive thresholding, segmentation, and contour extraction, in addition to filtering. After finding marker candidates, the inner white binary codification is checked by dividing the marker into cells. The cells are counted and extracted, and the black and white bits in the cells are used to determine if the cell bit is black or white. The bits are analyzed, and it is determined if the marker belongs in the specific dictionary. This process detects the marker. For pose estimation, the calibration parameters of the camera must be known. A single pose can be estimated for either a single marker or with several markers. (OpenCV, n.d.).

3.1.4 Camera calibration

When taking a picture or video with a camera, distortion can occur. There are two major types of distortion, radial and tangential. Radial distortion causes straight lines to appear curved in the image. Tangential distortion can cause areas in the image to look closer than expected. Camera calibration can be used to undistort an image. To undistort the image, the distortion coefficients and the camera's extrinsic and intrinsic parameters are required. OpenCV (Open-Source Computer Vision Library) explains how camera calibration works in

(*Detection of Aruco Markers*, n.d.).

Intrinsic parameters are specific to the camera. These are parameters such as the focal length or the optical center of the camera. These can be used to create a camera matrix that can be used to remove distortion due to the lens of the camera. Extrinsic parameters correspond to rotation and translation vectors which translate the coordinates of a 3D point. The distortion coefficients and the intrinsic/extrinsic parameters can be found through camera calibration. The calibration is done by holding a checkerboard up to the camera while running a camera calibration program. A checkerboard is used because it has many corners. Alternatives are a ChArUco (Chess + ArUco) board or a circular grid.

The camera calibration program takes in the 3D object points from the checkerboard and the 2D image points in the image. It gets these points by using a corner finding algorithm. The program gives out a camera matrix, distortion coefficients, and translation and rotation vectors. After these parameters are given out, the image can be undistorted. After having done camera calibration, the re-projection error can be found to check how good the calibration is. The closer to 0 the re-projection error is, the better. A good re-projection error is said to be under 1, even though this depends on the camera. The re-projection error can be found by projecting the 3D points from the checkerboard to the 2D image points using the parameters obtained by the calibration and calculating the absolute norm between this projection and the corner finding algorithm.

Often, the camera calibration images are taken one by one with the checkerboard at different angles and distances. The images must be evaluated before they are added to the calibration. If the images decrease the re-projection error, they are added to the calibration. ROS has a camera calibration package that makes the calibration easier for the user by continuously taking samples and evaluating the calibration.

In the simulations, camera calibration is not necessary. This is because there is no distortion in the simulations. Here, the calibration matrix is already defined in the files that describe the simulated camera. When moving to a physical setup, camera calibration is necessary. This is both to deal with distortion and because the ArUco detection package does not work without a calibration matrix.

3.1.5 LiDAR

An important tool that will be used later in this thesis is the LiDAR sensor. A LiDAR sends out laser rays and is a remote sensing method based on the backscattering of light. This means that it uses the reflection of the laser rays that hit something and travel back to the direction they came from. The LiDAR has a laser source and a receiver that accepts the laser reflection. By comparing the time delay and frequency shift in the emitted and reflected light, the distance and movement of an object hit by the LiDAR sensor can be determined

(Holtet, 2018). The output of a LiDAR can be a point cloud or a laser scan. The point cloud is a 3D point representation of the LiDAR data, while a laser scan is a 2D point range representation of the LiDAR data. The default data representation of the LiDAR used in this thesis is laser scan data.

3.1.6 Moving Average Filter

In this thesis, a moving average filter will be applied to the velocity of the follower to reduce noise. The moving average filter operates by averaging some points from the input signal to produce each point in the output signal. In equation form it looks like this:

$$\tilde{v_i} = \frac{1}{N} \sum_{j=0}^{N-1} v_{(i-j)} \tag{1}$$

where v is the input and $\tilde{v}$ is the filtered output.

The moving average filter is very simple yet very effective. It is optimal for reducing random white noise while keeping the sharpest step response. The moving average filter is a good smoothing filter but a bad low pass filter because of its frequency response. (Smith, 1999).

Something to note with the filter is that since it uses contributions from the previous velocities, the system puts less weight on the current calculated velocity. This introduces a delay because the filtered velocities are based on an arbitrary combination of previous velocities in addition to the current velocity. The delay could make the reaction time of the system slower. This is because instead of putting all weight on the current information, the system now uses a combination of old and new information.

3.1.7 Hough Circle Transform

The Hough Circle Transform is a deviation of the Hough Transform. The Hough Transform is used to isolate features of a particular shape in the image. The Hough Circle Transform is specialized for circles. This is the detection method used in (Miao et al., 2021), where they use it to detect a red ball. The Hough Circle Transform can be used to determine the parameters of a circle when several points that fall on the perimeter of a circle are known. An image containing many points will have some of these points falling on the perimeter of circles, then the detection program must find the center and radius to describe each circle

(Rhody, 2015). After detecting the circles, the radius of each circle and their center points in the image can be obtained in pixel coordinates.

3.1.8 ROS and ROS packages

ROS Noetic is used in the simulations and physical setup. It is a set of open-source software libraries and tools that can be used to build robot applications (Open Robotics, n.d.). ROS is essential, as it is an Operating Software that has everything needed for the robots to be described and used. All communication between the components and sensors of a robot is handled by ROS, and it has visualization and more. ROS has many useful packages that are going to be used in this thesis. Packages contain nodes, launch files, and most also have parameters that can be chosen. ROS also has tools like messages and topics, which let nodes and packages use information from the components of the robots and other nodes and packages. The packages that are used will be described in this section.

In the simulations, a custom marker model had to be created by making a square object and setting the texture of the object as an image of the ArUco code. The gazebo_link_attacher package can be used to attach the marker model to the back of the leader.

The TF2 package can be used to keep track of multiple coordinate frames and see how they evolve over time. The package maintains the relationship between the frames in a tree structure called a transformation (TF) tree. The TF2 package allows the user to get the transformation between any two frames directly if they are connected in the TF tree. The transforms are composed of the translation and rotation between the different frames. The package can, for example, give the transformation between the base frame of the robot and the camera frame.

While the TF2 package gives the transformation and relation between the links and coordinate frames of the robots over time, the frames must first be defined somewhere. The robot_state_publisher package publishes the state of the robot frames to the TF2 package. This means that the robot_state_publisher takes some description of the robot and its frames and publishes this to the TF2 package. Publishing means sending information to a topic. The robot_state_publisher requires a description of the robot and its coordinate frames. When using a custom robot, this description can be custom-made to suit the robot links and joints. The description of the Turtlebot is already given in the turtlebot_description package.

The aruco_detect package is a sub-package of the fiducials package; it contains a detection node with several parameters. If the aruco_detect package sees a marker with a predefined ID and size, the marker is detected. If the ID and size parameters that are set for the aruco_detect node do not match the ID and size of the ArUco code, the marker will not be detected. When the aruco_detect package detects a marker, it detects the corners of the

marker and gives out a transformation between the follower's camera frame and the marker frame. With the transformation between the marker and camera frames, the TF2 package can be used to get the transformation between the follower's base frame and the marker frame. The aruco_detect package also gives the image position of the corners of the marker in pixel coordinates.

The camera_calibration package makes calibrating the camera easier. The package has a camera_calibration node. Running this node and holding a checkerboard up to the camera at different distances and angles makes the node store new samples. The node shows how well the calibration is going while taking the samples and what kind of samples are needed by showing a bar that fills up depending on the type of samples that are taken. For example, there is a bar showing if there is enough variety in the checkerboard distances of the sampled images. When the bar is full for one specific type of sample, it means the node has enough variety in the samples of this type. After the calibration is done, the package gives out a camera calibration matrix. The package also has a node to check the re-projection error of the calibration. The error is checked by running a calibration_check node while keeping the checkerboard in view of the camera; the node gives out the error.

The camera_calibration package calibrates the camera and gives the camera calibration matrix, but it does not rectify/undistort the image. To undistort the image, the image_proc package must be used with the calibration matrix. This package handles image processing and gives a rectified image out, which can be used instead of the raw image. The processed image can, for example, be used by the aruco_detect package.

The pointcloud_to_laserscan package takes a 3D point cloud and transforms it into a 2D laser scan. It can also go the other way by transforming a 2D laser scan into a 3D point cloud, which can be used for operating with points instead of laser scan data. Operating with points is useful because raw laser scan data can be hard to transform and perform other operations on, and it can be easier to work with points.

Gazebo_ros_pkgs is a set of packages that provides everything required to run robot simulations in Gazebo. The package also integrates ROS and uses tools provided by ROS, such as ROS messages.

There is a Turtlebot3 package that has all the necessary descriptions, nodes, and scripts to use the Turtlebots in simulations and on a physical setup. This package handles LiDAR drivers, Turtlebot messages, and all other basic functionality of the Turtlebot. When using the Raspberry Pi camera module, this package also contains everything required to make use of the camera module. In the case where a USB camera is used, the cv_camera package has to be used for the camera instead.

Documentation of the gazebo_link_attacher package can be found in (PAL Robotics, n.d.). Documentation of the other packages can be found in (*ROS Wiki*, n.d.). When using ROS, nodes and packages can be put into a .launch file to initialize all the necessary nodes

and packages for the experiments simultaneously. Many packages also come with parameters that can be changed or tuned, such as the arcuo_detect package allowing users to change detection parameters. When including a node or package, the parameters can be changed in the .launch file.

3.1.9 Topics, messages, and nodes

Topics and messages can be used to transfer information within the robot system, for example, from sensors and frames to the robot or a node to the wheels of the robot.

As mentioned, a robot has coordinate systems that are connected in a TF tree in the TF2 package. Some packages publish, describe, or make new frames and then publish them. An example is the aruco_detect package, which defines a marker frame and publishes it while the marker is detected. The transforms and frames are being written and updated by publishers through topics. Topics are communication channels where messages can be sent and received. The messages that are sent and received need to have a format that is specific to the topic. Topics can be published to, meaning that something is sending information to the topic, or subscribed to, meaning that something is waiting for messages on the topic and reading the messages. Custom transforms and frames can be added if needed. This can be done by publishing a new frame and a transformation between the new frame and an existing frame.

Nodes can subscribe and publish to topics. A node is a program that can be used to send or receive information to and from topics and use this information to perform operations. For example, a node can subscribe to the /scan topic, which gives information about the LiDAR sensor and what it is detecting. It can then transform the LiDAR data from the base_scan frame to the base_link frame using the /tf topic, which gives the transforms between frames. After this, it can calculate a velocity based on the sensor information and then publish the velocity to the /cmd_vel topic, which controls the wheels of the robot. In the same way, the TF2 package receives information from nodes and packages to keep track of the transforms between the different frames of a robot.

Figure 3.2 shows a TF tree of the simulated leader Turtlebot. The world frame is connected to the odom frame of the Turtlebot. The odom frame is connected to the base_footprint, and the base_footprint is connected to the base_link frame. The base_link frame is connected to the other frames of the Turtlebot, which is why it is used as the base frame of the robot. In this figure, only half the TF tree is shown. The other half is the TF tree for the follower Turtlebot. The two Turtlebots are connected through the world frame. In the physical setup, the two Turtlebots are not connected through a world frame; their TF trees are separate. When a marker is detected, its frame shows up as fiducial_0 and is connected under the camera_rgb_optical frame of the follower. In the physical setup, the camera_link of the follower and the frames connected under it are replaced by the custom USB camera

frame. The TF tree also shows the broadcaster, which is the topic that the transformation is published in.

3.1.10 Gazebo and RViz

Gazebo is an open-source 3D robotics simulator that simulates real-world physics. It allows for testing and control of robots in simulations before implementation of the controllers on a physical setup. Gazebo is designed to reproduce dynamic environments that the robots may encounter. All simulated objects in Gazebo have mass, friction, velocity, etc. to let them behave realistically when pushed, pulled, etc. The robots in Gazebo are dynamic structures composed of rigid bodies connected via links and joints. Linear and angular forces can be applied to the joints to make them move. (Koenig & Howard, 2004).

The environment in Gazebo is described by a .world file. This file contains the simulated world, and the world can be populated with different models such as boxes, squares, and more complicated custom models, such as robots. The models are described by URDF and SDF files. These files can give the description of robots and how the different links and joints of the robots are attached and should behave, the kinematics of the robots, the frames, etc.

RViz is a program similar to Gazebo where the robot can be modeled, and simulations can be done. RViz is used less for simulations and more for debugging. This is because RViz can visualize information given in topics, such as the laser scan data given by the LiDAR in the /scan topic or the different coordinate frames given by the /tf topic. This is useful to ensure that everything in the setup is as it should be, that the ArUco marker is being detected properly, that custom transforms and frames are set properly, etc.

3.1.11 Turtlebot3 frames

When implementing controllers on a robot, it is important to know where the links and joints are relative to each other to be able to transform data or do other types of transforms. The frames are kept track of by the /tf topic, but not all the transforms are needed. In this section, some of the important frames, links, and their location will be mentioned.

The base_footprint frame of the Turtlebot3 is located between the wheels and is on the ground. The base_link frame, which is considered the base frame of the robot, is also located between the wheels. It located is 0.01 meters above the base_footprint. Since the base_link frame is considered the base frame of the robot, the sensor data must be transformed to the base_link frame. This is so all data has a common frame of reference for calculations and comparisons. The base_scan frame is located behind the base_link frame, a little above it, directly on the Turtlebot3 LiDAR. This is the reference frame for the LiDAR data. To

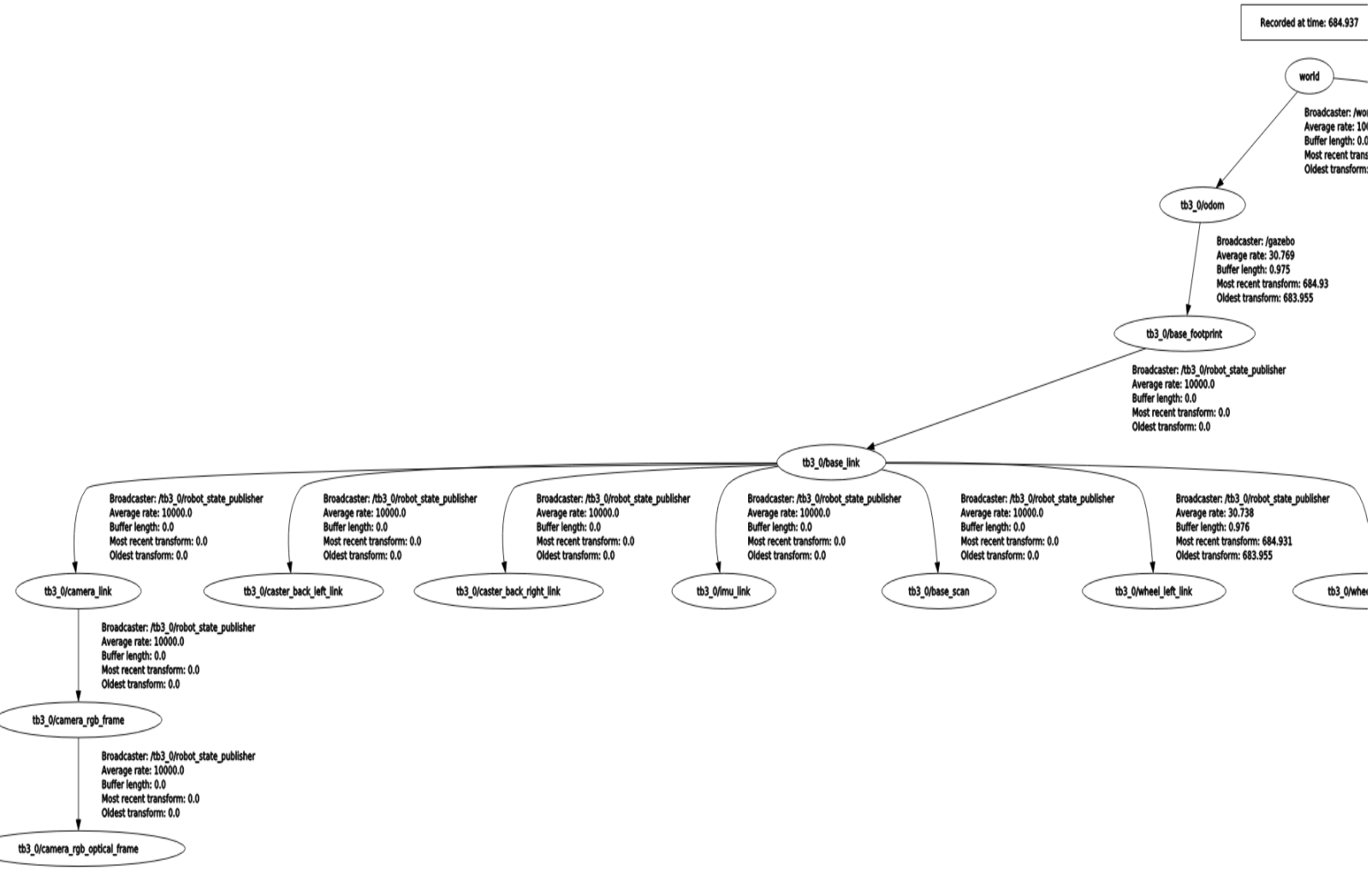

Figure 3.2: A figure of the TF tree of the leader robot in the simulations. The TF tree gives information about the connection between frames, which topic is broadcasting the transform, and the rate the transform is being broadcasted at.

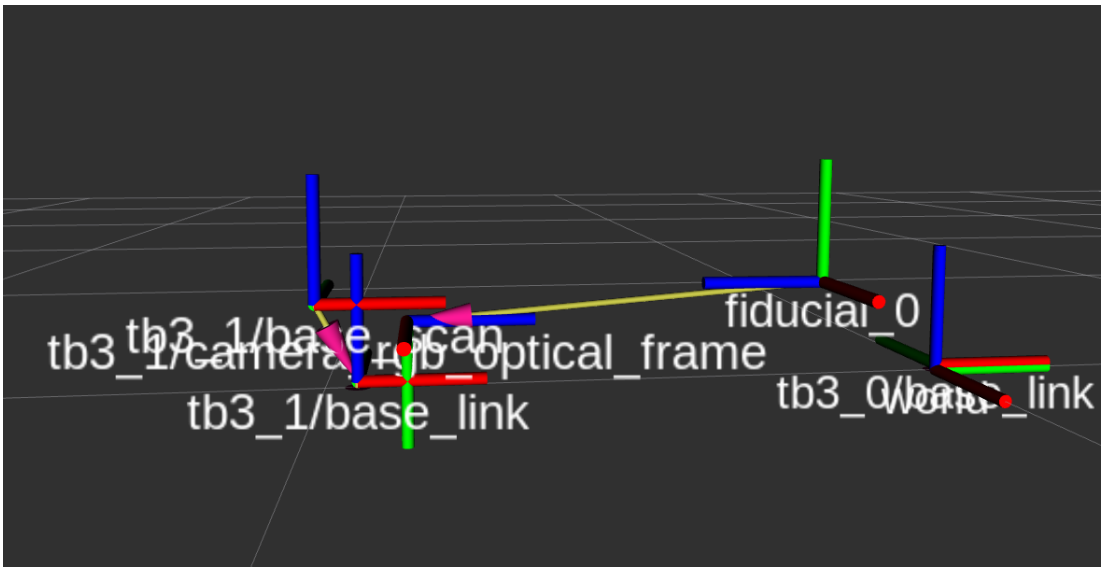

Figure 3.3: A figure of different frames of the robots in the simulations. The follower is standing behind the leader while detecting the marker. This shows the relation between the most important frames for control in this thesis.

transform LiDAR sensor data to the base frame of the robot, the transform between the base_scan and the base_link is used.

The Raspberry Pi camera has several frames. The important one is the camera's optical sensor (camera_rgb_optical) frame. It is important to note that this is the frame of the image sensor of the camera. This frame is also referred to as the camera frame and was shown in figure 3.1. The base_link, footprint, and scan frames all have the same orientation, with the x-axis pointing forwards. The camera frame is rotated relative to the base frame, so the z-axis is pointing forward and the y-axis downwards. The camera sensor data is given in the camera frame. To transform camera data to the base frame of the robot, the transformation between the camera frame and the base_link frame is used.

In the physical setup, a USB camera is used instead of the Raspberry Pi camera. Since the USB camera is not part of the Turtlebot3 description, it has no predefined frame or transformation between it and the base_link. This means that a custom transformation and frame have to be made by publishing the information. For this, a custom camera frame that emulates the camera_rgb_optical frame of the Raspberry Pi is made. Since a USB camera is different from the Raspberry Pi, the custom transformation between the camera and the base_link frame is slightly different than in the Turtlebot3 description.

Figure 3.3 shows the relationship between some of the frames of the follower, as well as the base_link of the leader. The fiducial_0 frame is the frame of the marker. The marker frame is pointing to the camera_rgb_optical frame since this is the frame its pose is given

in relation to. This image was taken while the robots were standing still in their initial positions in the simulations, while the marker detection was running. The base_footprint frames have not been included. If they were, they would be directly under the base_link frames of the robots with the same orientations.

3.1.12 Unicycle model

The model used in both the chosen control methods is the unicycle model. The unicycle model is used on systems with two wheels on an axle, like the robots in this thesis. If a system with four wheels on two axles is used, the bicycle model is more appropriate. Working with the movement and velocity of a differential drive robot with two wheels can be complicated. Therefore, a model like the unicycle model is preferred to simplify the problem. In the unicycle model, the robot is thought of as having a single wheel that can move with the desired velocity v and an orientation θ. The model can be used to calculate the velocities of the system (*The Unicycle Model*, n.d.).

The model is described as:

$$\begin{aligned} \dot{x} &= v\cos\theta \\ \dot{y} &= v\sin\theta \\ \dot{\theta} &= \omega \end{aligned} \tag{2}$$

where (x, y) is the position, v is the velocity, ω is the angular velocity, and θ is the orientation.

3.1.13 Motion capture

Motion capture is the process of digitally tracking and recording the movements of objects in space. The motion capture system uses sensors to detect motion, processes the data, and then stores the motion data.

There are different types of motion capture. Acoustic motion capture calculates the position based on time of flight, which measures the time between sending a signal from a transmitter and the signal being picked up by a receiver. There is mechanical tracking, where a mechanical construction measures the angle and distance between different mechanical parts. This is mostly used for tracking people by using an exoskeleton with sensors on it. There is also marker-based or marker-less optical tracking that uses cameras. In the marker-less tracking, everything in the FoV of the cameras is detected. Segmentation and

filtering are then used to get blobs. The blobs can be compared to predefined models of objects that should be tracked, such as a human body.

In this thesis, marker-based optical motion capture with passive reflective markers is used. Here, spherical markers are attached to the robots in patterns. A fixed pattern of several markers can be used to identify rigid bodies. At least three spherical markers are needed to define a rigid object, even though having more can make the tracking more robust in case some markers are not detected. Passive markers are markers that reflect light from external sources. There are also active markers that emit light. (Nymoen, 2013)

There are several cameras attached to the walls of the lab. All cameras point to the area where the tracking should be done from different directions and angles. Each camera produces a 2D black image where the spherical markers are detected as white pixels. To get a reading of the pose of the rigid bodies, multiple cameras must have a clear view of the markers that define the rigid body.

3.2 Distance-and-heading-based method

In (Verginis et al., 2015), they use multiple followers, and their equations are set up for more than one follower. In this thesis, there is only one follower. This simplifies most of the equations and models, making equations that used to be vector/matrix equations into normal equations. The model they use for the vehicles is the unicycle model:

$$\left.\begin{aligned} \dot{x}_i &= v_i \cos\theta_i \\ \dot{y}_i &= v_i \sin\theta_i \\ \dot{\theta}_i &= \omega_i \end{aligned}\right\}, i = 1, \ldots, N \tag{2}$$

where (x_i,y_i) is the position of vehicle i. θ_i is the orientation. $N = 2$ since there is a leader and one follower. v_i and ω_i are the linear and angular velocities. They introduce $d(t)$ and $\beta(t)$, which are the relative distance and bearing angle between the two vehicles. The bearing is the angle between the follower's forward direction and the leader.

They define the desired distance between the two vehicles as d_{des}. The goal is to make $d(t) - > d_{des}$ and $\beta(t) - > 0$. So the desired position of the follower is directly behind the leader at a distance d_{des} with the follower pointing at the leader. They define some constraints for the distance: $d_{col} < d(t) < d_{con}$. d_{con} is the maximum distance; the follower cannot detect the leader if the relative distance is greater than this. d_{col} is the minimum distance; if the relative distance is smaller than this the follower cannot detect the leader, and the robots risk colliding. They introduce similar constraints for the bearing between the two vehicles: $|\beta(t)| < \beta_{con}$. β_{con} is the bearing where the leader stops being detected by the

follower.

The constraints must be taken into account when initializing the system so that they are not violated at initialization. Figure 3.4 shows a model of the leader and follower with the constraints and frames. The leader is detected by having a marker attached behind it that the follower can use to find the relative distance and bearing.

The distance between the two vehicles is defined as:

$$d(t) = \sqrt{(x_f(t) - x_l(t))^2 + (y_f(t) - y_l(t))^2}$$

where $x_l(t)$,$y_l(t)$,$x_f(t)$,$y_f(t)$ are the respective x and y positions of the leader and follower in the world frame. The follower's frame is used as a reference frame instead of the world frame so that the position of the leader is relative to the follower. Using the follower's frame as a reference means that $x_f(t)$ and $y_f(t)$ can be set to 0.

With the previously given parameters, they define the distance and heading errors of the follower as:

$$\left.\begin{array}{c} e_d(t) = d(t) - d_{des} \\ e_\beta(t) = \beta(t) \end{array}\right\} \tag{3}$$

It is possible to differentiate equation (3) with respect to time and insert equation (2) to get the evolution of the error. This is not required for the control of the system, so it is not presented here.

After defining the errors, the controller can be designed. The controller should achieve a predefined transient and steady-state response for $e_d(t)$ and $e_\beta(t)$ while avoiding violation of the previously defined constraints. In Section III of (Verginis et al., 2015), they say prescribed performance characterizes behavior where the errors evolve over a predefined region that is bounded by absolutely decaying functions of time. The prescribed performance is given by the inequalities:

$$-\underline{M}_j \rho_j(t) < e_j(t) < \overline{M}_j \rho_j(t) \tag{4}$$

for all $t \geq 0$.

The subscript: $j \in \{d, \beta\}$ is used to reduce the number of equations. When using j, it means that the equation is the same for both d and β. The subscript is introduced to avoid showing the same equation twice with different indexes.

$\underline{M}_j$ and $\overline{M}_j$ are positive parameters that are selected appropriately to satisfy the collision and connectivity constraints represented by d_{con}, d_{col} and β_{con}. $\rho_j(t)$ is the steady-state performance and is defined as:

$$\rho_j(t) = (1 - \frac{\rho_{j,\infty}}{\max\{\underline{M}_j, \overline{M}_j\}})e^{-l_j t} + \frac{\rho_{j,\infty}}{\max\{\underline{M}_j, \overline{M}_j\}} \tag{5}$$

It is a smooth, bounded, and decreasing positive function of time. It is always between 1 and 0. l_j is the desired transient time constant, and $\rho_{j,\infty}$ is the maximum steady-state error. The value chosen for l_j depends on how fast the system should converge. The value of $\rho_{j,\infty}$ depends on what is an acceptable desired steady-state error. In (Verginis et al., 2015), they say $\rho_{j,\infty}$ can be chosen as an arbitrarily small number. $\underline{M}_j$ and $\overline{M}_j$ are selected as:

$$\left.\begin{array}{l} \underline{M}_d = d_{des} - d_{\text{col}} \\ \overline{M}_d = d_{\text{con}} - d_{des} \\ \underline{M}_\beta = \overline{M}_\beta = \beta_{\text{con}} \end{array}\right\} \tag{6}$$

After defining the performance function $\rho_j(t)$ and selecting the positive parameters $\underline{M}_j$ and $\overline{M}_j$, the decentralized control equations can be set up.

They first define a normalized error:

$$\xi_j(e_j,\ t) = \frac{e_j}{\rho_j(t)} \tag{7}$$

then the control equations can be written as:

$$v(\xi_d,\ t) = k_d \epsilon_d(\xi_d)$$

$$\omega(\xi_\beta,\ t) = k_\beta(\rho_\beta(t))^{-1} r_\beta(\xi_\beta)\epsilon_\beta(\xi_\beta) \tag{8}$$

By inputting the calculated control velocities into the follower's wheels, the follower can be controlled. The controller of the system is divided into a linear velocity controller and an angular velocity controller. k_d and k_β are positive control gain variables that must be tuned.

$r_\beta(\xi_\beta)$ is defined as:

$$r_\beta(\xi_\beta) = \left[\frac{\frac{1}{\underline{M}_{\beta_i}} + \frac{1}{\overline{M}_{\beta_i}}}{(1 + \frac{\xi_{\beta_i}}{\underline{M}_{\beta_i}})(1 - \frac{\xi_{\beta_i}}{\overline{M}_{\beta_i}})}\right]$$

$\epsilon_j(\xi_j)$ is defined as:

$$\epsilon_j(\xi_j) = \ln\left(\frac{1+\frac{\xi_{j1}}{M_{j1}}}{1-\frac{\xi_{j1}}{M_{j1}}}\right)$$

With this, everything that is needed to implement the controller has been presented. If the normalized error ξ_j becomes so big that $1-\frac{\xi_{j1}}{M_{j1}}$ becomes negative, $\epsilon_j(\xi_j)$ becomes undefined. This occurs when the error goes outside of the boundary/prescribed performance described by equation (4) and causes the system to become unstable.

The term bearing (angle) will be used when talking about the relative angle between the follower and leader. The term heading error will be used when talking about the follower's angular error. From now, this control method will be referred to as the distance-based method.

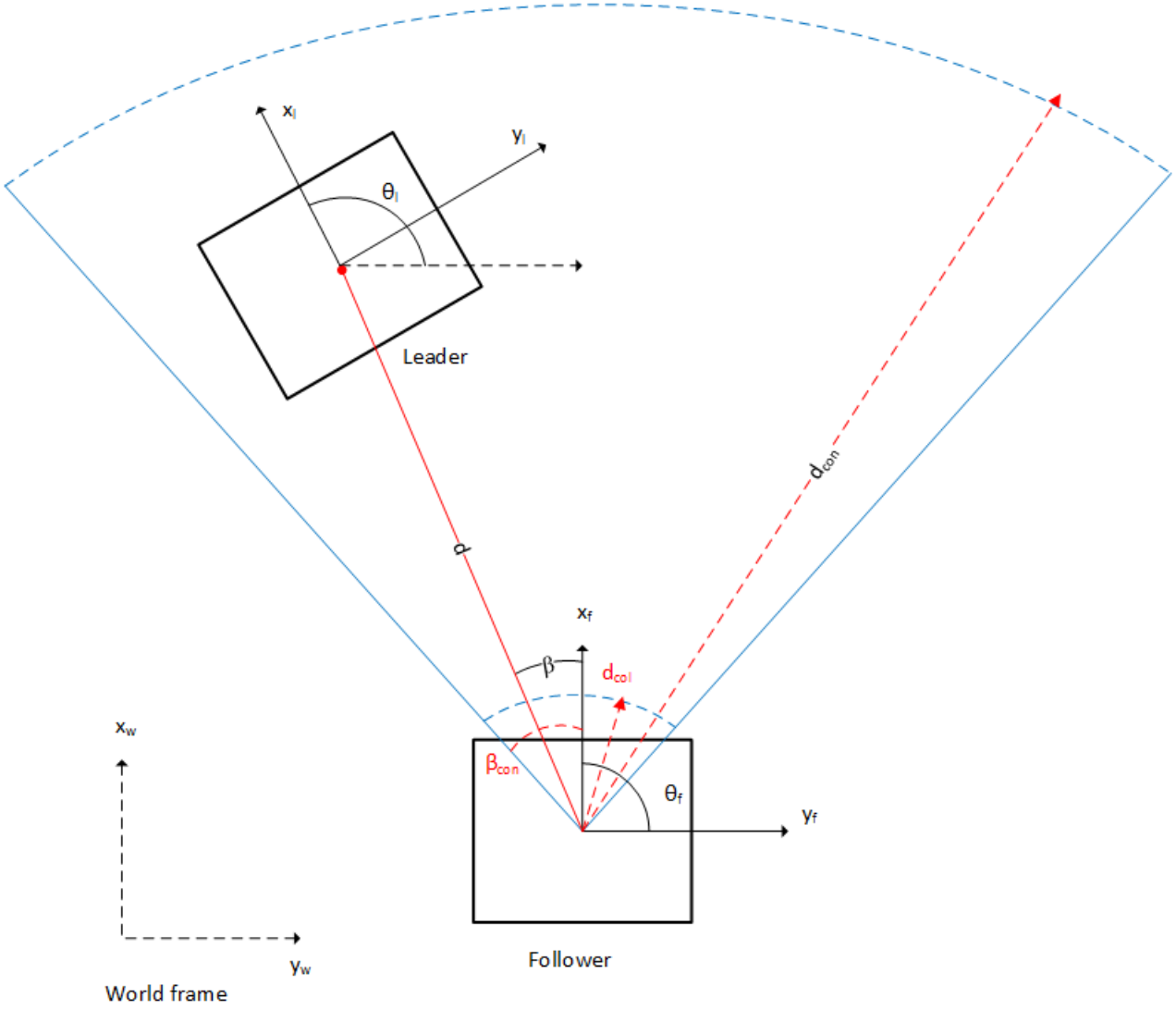

Figure 3.4: Model of the leader and follower as seen from above. This shows the different constraints, as well as the frames of the two robots. The x-axis of the robots points in the direction they are facing. The controller should keep the distance and bearing angle between the two robots within the constraints. Image inspired by Fig.1 in (Verginis et al., 2015).

3.3 Pixel-based method

The second controller that will be implemented is given in (Miao et al., 2021). As previously mentioned, the paper presents two controllers. The Nussbaum gain adaptive controller and the static nonlinear gain controller. Only the static nonlinear gain controller will be used in this thesis, as it was the controller that gave the best results in (Miao et al., 2021). In this section, only the most relevant aspects to the implementation of the controller will be presented.

In (Miao et al., 2021), they start by defining the unicycle model for the robots as shown in equation (2) in section 3.2. $\boldsymbol{r_i} = [x_i, y_i]^T$ and θ_i are the position and orientation of the robots in the world frame. The motion of the leader in the follower's frame can be described as:

$$\boldsymbol{r_{lf}} = \begin{bmatrix} \cos\theta_f & \sin\theta_f \\ -\sin\theta_f & \cos\theta_f \end{bmatrix} (\boldsymbol{r_l} - \boldsymbol{r_f}). \tag{9}$$

where $i \in \{l, f\}$ is the subscript of the leader and follower respectively. The equation can be derivated to obtain the relative kinematics, which is not presented here since it is not required for the control of the system.

The follower has a camera where the z-axis of the camera frame is parallel to the x-axis of the follower's base frame, as shown in figure 3.3. This means that the camera's optical axis is pointing in the same direction as the follower. The controller requires a feature point $\boldsymbol{P}$ on the leader to be able to detect and follow the leader. In (Miao et al., 2021), this feature point is given by a colored ball attached to a pole on the leader; the camera detects the ball by using the Hough Circle Transform. Their camera is attached to a pole on the follower.

The coordinates of the feature point $\boldsymbol{P}$ in the world frame transformed to the camera frame are given as:

$$\boldsymbol{P} = (X, Y, Z)_{world} = (-y_{lf}, h, x_{lf})_{camera} \tag{10}$$

where h is the height between the camera's optical center and the feature point, as shown in figure 3.1. h is a constant, and it is also referred to as the feature depth parameter. While h is defined and known, it is assumed to be unknown and is not used in the control equations. x_{lf} and y_{lf} are the x and y position of the leader relative to the follower's frame.

According to the pinhole model of the perspective camera, the pixel coordinates $\boldsymbol{s} = (m, n)$ of $\boldsymbol{P}$ can be found as:

$$
\begin{aligned}
n =& \alpha_n \frac{Y}{Z} + n_0 \\
m =& \alpha_m \frac{X}{Z} + m_0
\end{aligned} \tag{11}
$$

where α_n and α_m are constant scaling factors that are larger than 0. n_0 and m_0 are the pixel coordinates of the camera's principal point (projective center). The pixel coordinates n and m are given directly from the image if the feature point is detected by the camera.

In the paper, they also define the normalized image coordinates p and q, and the derivative of the pixel coordinates n and m. This is not necessary for the implementation of the static nonlinear gain controller.

The pixel coordinates are subjected to some FoV constraints. They define these FoV constraints as:

$$
n_{\min} \leq n \leq n_{\max}, \quad m_{\min} \leq m \leq m_{\max} \tag{12}
$$

where $n_{\min}$, $n_{\max}$ $m_{\min}$ and $m_{\max}$, are the minimum and maximum values of n and m.

The control objective is to make the pixel coordinates $\boldsymbol{s} = (m, n)$ converge as close as possible to some desired pixel coordinates $\boldsymbol{s_{des}} = (m_{des}, n_{des})$. It is assumed that the velocity of the leader is bounded and that the feature point $\boldsymbol{P}$ is initially in the FoV of the follower's camera. It is also assumed that $\boldsymbol{P}$ does not pass through the optical center of the camera, meaning that $h \neq 0$. h must not be equal to 0, as this leads to singularities in the control design.

They define pixel errors as $\boldsymbol{e} = [e_n, e_m]^T$, with

$$
e_n = n - n_{des}, \quad e_m = m - m_{des}. \tag{13}
$$

There is a prescribed performance which characterizes the evolution of the pixel errors. The prescribed performance is given by:

$$
-\underline{C}_k \rho_k(t) < e_k < \overline{C}_k \rho_k(t) \tag{14}
$$

The steady-state performance is given by:

$$
\rho_k(t) = \left(1 - \frac{\rho_{k,\infty}}{\max\{\underline{C}_k, \overline{C}_k\}}\right) e^{-l_k t} + \frac{\rho_{k,\infty}}{\max\{\underline{C}_k, \overline{C}_k\}} \tag{15}
$$

where $k \in \{n, m\}$. $l_k > 0$ and $\rho_{k,\infty} > 0$ and represent the convergence rate/desired transient time constant and the maximum steady-state error respectively. $\underline{C}_k$ and $\overline{C}_k$ are given as:

$$\begin{aligned} \underline{C}_n &= n_{des} - n_{\min}, \quad \overline{C}_n = n_{\max} - n_{des} \\ \underline{C}_m &= m_{des} - m_{\min}, \quad \overline{C}_m = m_{\max} - m_{des}. \end{aligned} \tag{16}$$

The prescribed performance part is very similar to what was shown in the distance-based method.

Using the previous definitions, they define a error transformation function ε_k as:

$$\varepsilon_k = \ln\left(\frac{e_k + \underline{C}_k \rho_k(t)}{\overline{C}_k \rho_k(t) - e_k}\right). \tag{17}$$

After getting this equation, they take the derivative of it and use it to show that the control problem can be solved, which is not going to be presented here.

Finally, the control laws for the static nonlinear gain controller are given as:

$$[v_f, \omega_f]^T = [k_n M(\varepsilon_n), -k_m \varepsilon_m]^T \tag{18}$$

where k_n and k_m are the control gains, with $k_n \neq 0$ and $k_m > 0$. $M(\varepsilon_n)$ is a function that has properties such that ε_n is bounded. In the paper, $M(\varepsilon_n) = \varepsilon_n cos(\varepsilon_n)$.

Again, it can be seen that the controller is split into a linear velocity controller and an angular velocity controller. It can be seen from equation (17) that if e_k becomes larger than $\overline{C}_k \rho_k(t)$, the equation becomes undefined. This can cause the system to become unstable. It is therefore important that the error stays within the boundary/prescribed performance. From now, this method will be referred to as the pixel-based method.

Chapter 4

Methods

In this chapter, additions and improvements to the control methods that were made in an attempt to make the controllers perform better, be more robust, or be more stable will be shown.

The additions in this chapter consist of adding a filter to reduce noise, trying a LiDAR sensor instead of using the camera, and then finally combining the LiDAR and filter with the original camera version of the control methods to hopefully get an improved version.

4.1 Markers

In (Verginis et al., 2015), a white spherical marker is attached behind the leader to find the relative distance and bearing between the leader and follower. Their marker detection method is not described. In (Miao et al., 2021), a red ball is attached to a pole on the leader, and the detection is done by using the Hough circle detection according to one of their earlier works: (Lin et al., 2021). In (Miao et al., 2021) they show results from both simulations and physical implementation, while in (Verginis et al., 2015) they only show simulation results.

In this thesis, an ArUco marker is attached behind the leader. As mentioned in section 3.1.8, the aruco_detect package gives the transformation between the marker and camera frames when the marker is detected. Using the TF2 package, the transformation and relative pose between the base_link and marker frames can be found. This is used to find the relative distance and bearing between the follower and marker. The aruco_detect package also gives the pixel coordinates of the corners of the marker, which can be averaged to get the center of the marker. The center of the marker is used as feature point $\boldsymbol{P}$. The benefit of using ArUco detection is that it finds the pose of the marker relative to the camera frame, as well as the pixel position of the marker in the image. This makes it so the same marker can be

used in both control methods, despite them using different information. With Hough circle detection, for example, only the pixel position of the center of a detected circle would be found, and then additional techniques would have to be used to find the relative distance and angle.

An issue that became apparent in early simulations of the system was that the camera measurements could be noisy and have measurement error spikes in situations where the marker was not easily detectable. Early in the simulations, the marker's margin, which is the white border around the marker, was 1 cm. Having a margin this small caused issues with detection, such as noisy measurements, wrong measurements, and the marker not being detected at all sometimes. The marker could be in perfect view of the follower. Despite this, the detection would fail at times. If the marker stops being detected, the system keeps calculating the velocities based on the previous measurements. When the marker is detected again after some time, the measurements are updated, and this leads to a spike in the measurements and error. If the error spikes too much, the system becomes unstable because the error goes outside the boundaries set by the prescribed performance.

Later in the simulations, the marker detection was made more robust by increasing the margin of the marker to 3 cm. This made the measurements considerably less noisy and more accurate. It also made the detection more reliable, fixing the issue of the marker not being detected at times. This is because the larger margin makes it easier for the detection package to detect the corners of the marker, while the smaller margin made detecting the corners harder. Since the earlier simulations showed that using only a camera for control could become a problem in some situations, alternatives were tested.

4.2 Filtering

The first addition to the control methods is a filter to reduce the noise of the system. The aruco_detect package has disturbances and uncertainty that create noise. The marker frame can rotate and shift a little, or the pixel positions of the corners of the marker can have some uncertainty, which causes measurement noise. In addition, the system can have noise from other factors. This mostly applies in the physical setup. Examples of these are camera shaking, velocity spiking because of bad measurements, bad lighting, which makes marker detection worse, etc. Camera shaking will be used to refer to the camera shaking or moving a little up or down for a moment when the robot is driving, especially over uneven flooring such as tiny bumps or tile changes. In addition to the aforementioned sources of noise, there could also be unknown sources of noise.

This makes it so the measurements that are used to calculate the velocities contain noise, making the calculated velocities have more variation than they should. To remove some of the noise, a moving average filter has been implemented. This filter, as mentioned in section

3.1.6, generally has the form:

$$\tilde{v_i} = \frac{1}{N} \sum_{j=0}^{N-1} v_{(i-j)} \tag{1}$$

Where v is the input and $\tilde{v}$ is the filtered output. A simple 1st-degree version of the moving average filter can be written as:

$$\tilde{v_i} = av_{i-1} + bv_i$$

where a and b weights. v_i is the velocity the controller calculates based on the current measurements. v_{i-1} is the previous velocity of the system. $\tilde{v_i}$ is the filtered velocity. The average of the two velocities is taken when both a and b are set to 0.5. Instead of taking the average, a choice can be made to put more weight on the current time-step by setting b to a bigger number than a. The sum of the weights should always be equal to 1, and the individual weights should be between 0 and 1. In this thesis, a second-degree moving average filter is used as it removes more noise than the 1st order version. It looks like this:

$$\tilde{v_i} = av_{i-2} + bv_{i-1} + cv_i \tag{19}$$

The input in the filter is the current velocity calculated by the controller and the two previous velocities of the system. The filter is used on both the linear and angular velocities.

4.3 Adding and using a LiDAR

As mentioned in section 4.1, early simulations of the system had issues with marker detection due to the marker margin being too small. Before a larger margin was tested, a LiDAR version of the controllers was explored as an alternative to using the camera. Even though increasing the margin size made the marker detection more robust, a LiDAR-based version could still be useful in other situations where the marker detection is unreliable or in situations where the camera-based version of the controller has other issues.

A LiDAR version of the controllers would not need to detect markers. As a result of not having to rely on the camera and marker detection, the system should be more robust, as

it does not have noise or spikes in the measurements due to unreliable marker detection or other camera-related issues. LiDARs, as previously mentioned, work by sending out a laser scan in a radius and checking if the lasers detect an object. If the laser rays detect an object, the distance at which the object was detected and the bearing angle can be calculated. The distance is given directly, while the bearing can be calculated based on which laser rays hit an object. Depending on the resolution of the LiDAR, the angle calculations can be imprecise.

4.3.1 LIDAR measurements

When using the LiDAR, it is important to know how the robot is set up and how the different frames on the robot are placed relative to each other. This was shown in section 3.1.11. The LiDAR measurements are given relative to the base_scan frame of the robot, but all measurements should be given in the base_link frame, as this is the base frame of the robot. Two ways of transforming the data between the two frames were tested. The first is based on transforming the LiDAR scan into a point cloud, then doing the transformation on the point cloud and transforming back to a LiDAR scan. The second is based on trigonometry.

For the point cloud method, the pointcloud_to_laserscan package is used. The package takes the laser scan data given by the LiDAR and transforms it into a point cloud. The point cloud data consists of a set of points in space given in the base_scan frame. A transform can be done on the point cloud by using the TF2 package. This transforms the point cloud from the base_scan frame to the base_link frame. The transformed point cloud can be transformed back to laser scan data by using the pointcloud_to_laserscan package again. This gives the laser scan data in the base_link frame.

This method worked, but not well enough to be used. Sometimes, the measurements from the transformed LiDAR scan data would be wrong, or the measurements would have spikes. This could lead to the system becoming unstable. The reason for these spikes could be because of the transformations that are done. When the laser scan data is transformed to a point cloud and then back to a laser scan, some information could be lost, or erroneous information could be added. The laser scan consists of 2D points and is a line of points. The point cloud consists of 3D data points in space. It could be that the transformation from 2D to 3D and back to 2D causes the aforementioned issues.

The second method of transforming data from the base_scan to the base_link frame uses trigonometry and is based on transforming the calculated distance and bearing measurements instead of the raw laser scan data. The relative distance between the base_scan frame and the detected object can be viewed as the hypotenuse in a right triangle, denoted by $\tilde{d}$. The angle in the triangle is the bearing measured in the base_scan frame, denoted by $\tilde{\beta}$. The base_link frame is 0.064 meters in front and 0.1 meters below the base_scan frame. Both frames have the same orientation, as seen in figure 3.3. Because the LiDAR scan data is 2D and does not consider height, the transformation between the two frames can be simplified

to a -0.064-meter translation along the x-axis of the base_link frame.

With the relative distance between the follower's base_scan frame and a detected object as the hypotenuse, and the bearing between them as the angle in the triangle, the right triangle can be decomposed into $\tilde{d}_x$ and $\tilde{d}_y$. This gives:

$$\begin{aligned} \tilde{d}_x &= \tilde{d} \cdot \cos(\tilde{\beta}) \\ \tilde{d}_y &= \tilde{d} \cdot \sin(\tilde{\beta}) \end{aligned} \tag{20}$$

Since the transformation between the base_scan and base_link is only a translation along the x-axis, it can be applied to $\tilde{d}_x$ as:

$$d_x = \tilde{d} \cdot \cos(\tilde{\beta}) + R$$

where d_x is the transformed $\tilde{d}_x$. R is the translation, which is -0.064 meters. The relative distance and bearing in the base_link frame can be found as:

$$d = \sqrt{d_x^2 + \tilde{d}_y^2} \tag{21}$$

$$\beta = \arctan \frac{\tilde{d}_y}{d_x} \tag{22}$$

where d and β are the distance and bearing in the base_link frame. The $\tilde{d}_y$ component remains the same in both frames, but the $\tilde{d}_x$ component is translated. In the simulation and implementation, the method using trigonometry is used to transform the LiDAR data from the base_scan frame to the base_link frame.

4.3.2 LIDAR in distance-based method

Implementing a LiDAR-only version of the distance-based method is straightforward. The information required for this method was the relative distance and bearing between the leader and follower. When the leader robot is in front of the follower, the LiDAR detects that there is an object there because some of the laser rays are reflected. Checking the measurements from only one laser ray can be noisy and unstable. Taking the average of all the lasers that detect something is more robust. The issue with taking the average is if the LiDAR detects additional objects. For example, if some of the laser rays detect a wall while some detect the leader, the average can become skewed.

To solve this, the measurements from the LiDAR can be thresholded. The threshold is made by using both the bearing and distance measurements and is based on the previous measurement. When the node starts, the first measurement is taken. All the individual laser ray measurements are checked. Based on the measurement from the laser rays, the maximum and minimum detected bearing angles are found, as well as the maximum detected distance. After getting the minimum and maximum measurements, thresholds are made based on them by adding a little to the measurement. The thresholds are set as:

$$t_\beta = max\ \beta + 3\ deg\ \Lambda\ min\ \beta - 3\ deg$$

$$t_d = d_{max} + 0.25 d_{max} \tag{23}$$

where t_β and t_d are the thresholds on the bearing and distance respectively. The thresholds are updated in the next iteration so that the current bearing and distance measurements are thresholded based on the thresholds made by using the previous measurements. If the measured distance or bearing is bigger or smaller than the threshold, the measurement is rejected. When using the previous measurements to threshold, the LiDAR must only detect the leader at the start so that the thresholds are initialized correctly.

Initially, the LiDAR starts with a 60-degree detection cone in front of it, with a maximum detection distance set at 2 meters. After thresholding, the cone becomes smaller, becoming ± 3 degrees bigger and $0.25d$ meters longer than the previous min/max bearing and max distance measurements from the object. Only detection within this cone will count. The detection cone adjusts depending on where the leader is relative to the follower, but the cone is restricted to the initial 60-degree detection cone

After thresholding the laser ray measurements and finding the average distance between the follower and leader, the average bearing can be found. This is done by checking which laser rays are detecting an object, finding the bearing angle of these, and taking the average over all of them.

The laser ray bearing measurements are found by:

$$\beta_{laser} = \theta_{min} + (i_{laser} \cdot \theta_{inc}) \tag{24}$$

The LiDAR has a minimum detection angle θ_{min}. Since a 360-degree LiDAR is used in this thesis, θ_{min} is 0 degrees while the max angle is 360 degrees. i_{laser} is the individual laser ray. θ_{inc} is the angle increment of the LiDAR, also called the angular resolution. With the

LiDAR used in this thesis, the angular resolution is 1 degree. The equation gives the bearing measurement for one laser ray, then the average of the calculated bearings can be taken.

With the relative distance and bearing between the two robots found using the LiDAR, they can be used in the distance-based method directly to calculate the control equations for the velocity.

4.3.3 LIDAR in pixel-based method

The pixel-based method requires the pixel coordinates m and n of a feature point instead of the relative distance and bearing angle. Equation (10) in section 3.3 shows the relationship between the point $\boldsymbol{P}$ in the world and camera frames. Equation (11) shows the feature point in pixel coordinates.

$$\boldsymbol{P} = (X, Y, Z)_{world} = (-y_{lf}, h, x_{lf})_{camera} \tag{10}$$

$$\begin{aligned} m =& \alpha_m \frac{X}{Z} + m_0 \\ n =& \alpha_n \frac{Y}{Z} + n_0 \end{aligned} \tag{11}$$

Using these equations, a LiDAR can be used to estimate the feature point $\boldsymbol{P}$ without using the camera. This can be done by finding the relative position of the leader and follower with a LiDAR. After the position has been found, the transformations in the equations can be used to estimate $\boldsymbol{P}$. $\boldsymbol{P}$ can then be used to calculate m and n, which can be used to calculate the control velocities of the follower.

To implement the LiDAR version of the pixel-based method, X and Z have to be found, meaning y_{lf} and x_{lf}. x_{lf} and y_{lf} are the relative x and y positions between the leader and follower. h, which was the height difference between the camera optical axis and the feature point, is known. Since the pixel coordinate of the marker center is used as $\boldsymbol{P}$, h can either be measured or found through a transformation between the marker frame and the camera frame. Y is therefore also known. m_0 and n_0 were the camera's projective center and could be found through the camera calibration matrix. α_m and α_n are constant scaling factors. Everything required to approximate $\boldsymbol{P}$ is known or can be found through the LiDAR, assuming that the camera was used to find some constants first, such as m_0 and n_0.

Trigonometry is used to create a right-angled triangle between the leader and follower positions, where the relative distance is the hypotenuse, and the bearing angle is the angle in the triangle. y_{lf} and x_{lf} can be found by decomposing the triangle.

This results in:

$$\begin{aligned} Z &= x_{lf} = cos(\beta) \cdot d \\ X &= -y_{lf} = -sin(\beta) \cdot d \end{aligned} \tag{25}$$

where β and d are the bearing and relative distance found by the LiDAR. With all the parameters obtained, they can be inserted into equation (11) to estimate the pixel coordinates m and n using only the LiDAR:

$$m = \alpha_m \frac{-sin(\beta) \cdot d}{cos(\beta) \cdot d} + m_0 = -\alpha_m tan(\beta) + m_0$$

$$n = \alpha_n \frac{h}{cos(\beta) \cdot d} + n_0 \tag{26}$$

Since the image coordinates have been estimated with the LiDAR instead of taken directly from the camera, this version might have more measurement errors than the camera version of the method. However, it should be more robust because there is no issue if the marker detection is unreliable. Other issues that could affect the camera, such as bad lighting or camera shaking, do not affect this version.

4.4 Combining LiDAR and camera measurements

Now that the additions to the distance-based and pixel-based methods have been shown, an idea could be to combine them to create a new version. As previously mentioned, many previous works combine LiDAR and camera measurements to great effect. The LiDAR is effective at tracking the relative distance between vehicles but less accurate at measuring the bearing angle. Using only the LiDAR might not give the accuracy of a camera-only version of the controller. The camera is good at measuring the relative distance and bearing but is less robust towards external factors and unreliable marker detection. Using only a camera might not give the stability and robustness of a LiDAR-only version. Combining the two sensors could make the system more robust and accurate. In addition to combining the two sensors, the moving average filter is added to remove noise.

From this, there are now four versions of both control methods. A camera-only version, which only uses the camera. A LiDAR-only version, which only uses the LiDAR. A filtered version that uses the camera and the filter. And a complete version, which uses the camera, LiDAR, and filter.

4.4.1 Combining measurements in the distance-based method

For the distance-based method, the implementation of the complete version is done by starting with the camera-only version. The camera-only version gets its measurements of the relative distance and bearing angle from the camera by detecting the marker. Then the LiDAR-only version, where the measurements of the relative distance and bearing are found from the LiDAR, is integrated with the camera-only version. The measurements from the two sensors are then fused. This is done by making a new measurement that consists partially of the measurements from the LiDAR and partially of the measurements from the camera and combining them:

$$Combined_m = w_1 * L_m + w_2 * C_m \tag{27}$$

$Combined_m$ are the combined measurements from the LiDAR measurements L_m and the camera measurements C_m. w_1 and w_2 are weights, the sum of them should be equal to 1, and the weights should be between 0 and 1. The new measurements are a weighted combination of the LiDAR and camera measurements, where w_1 and w_2 decide how much weight is put on the LiDAR or camera measurements. After combining the measurements, the filter is added, filtering the current velocities based on previous velocities, as shown in section 4.2.

4.4.2 Combining measurements in the pixel-based method

Implementing the complete version of the pixel-based method is very similar to how it was done in the distance-based method. The difference is that the measurements here are the pixel coordinates m and n instead of the relative distance and bearing. The measurements from the camera come from detecting the marker center in pixel coordinates. The LiDAR measurements come from estimating the pixel coordinates of the marker, as shown in section 4.3.3. The measurements are then fused like in the distance-based method. Finally, the filter is added.

Chapter 5

Experiments and results

In this chapter, the setup of the simulations, physical system, and experiments will be shown. Parameter choices, software setup, specifications of the equipment, and experiment results will also be presented.

5.1 Simulation Setup

Gazebo was used for simulation, and RViz was used for debugging the simulations and the physical setup. In the simulations, only one computer was needed. All the nodes and simulations were run by having several command terminals on the computer. The different command terminals run the simulation, ROS master, and the different nodes. In the Gazebo simulations, a world frame was placed in (0,0,0) as a reference between the two robots. To simulate the controllers, the leader was initialized in the origin of the world frame, and the follower was placed one meter directly behind the leader. When specifying that the follower was one meter behind the leader in Gazebo, the base_link of the follower was 0.8 meters behind the marker attached to the back of the leader. This is because the distance between the marker and the base_link of the leader is 0.2 meters.

In the simulations, it is possible to custom define the noise of the sensors. This was not done. The turtlebot_description package came with predefined noise for the simulated LiDAR to model the physical LiDAR. The simulated LiDAR had Gaussian white noise with a standard deviation of 10 mm, which matches the hardware specifications of the LiDAR given in section 5.2.5. The aruco_detect package had detection noise, as previously mentioned. The Turtlebots shook a little while driving in the simulations, this could cause noise, but it was not notable.

To run the simulations, a launch file was set up. The launch file launched the simulation

models of the Turtlebots, its sensors, and the world they were simulated in. It also initialized all the necessary nodes and packages, such as the ArUco detection, the transforms, etc. The simulated world was empty except for the Turtlebots. After the launch was completed and the Turtlebots were initialized in their positions in the world, the marker was attached to the leader using the gazebo_link_attacher package. After this, the simulations were ready, and the nodes that make the leader move and the follower follow the leader could be run.

5.2 Experimental Setup

This section will go over some of the details of the physical setup.

For the physical implementation of the system, two Turtlebot3 Waffle Pis were used together with two laptops that were used as remote PCs and a desktop computer that was used to control a motion capture system with a software called Qualisys Trackmanager. The motion capture system was a Qualisys system with 21 Miqus M3 cameras mounted on the ceiling and walls of the lab. The Turtlebots come equipped with a Raspberry Pi v2.1 camera module and a 360-degree LDS-01 LiDAR. In the physical implementation, there were issues with the Raspberry Pi v2.1 camera module and packages related to it. Because of this, a Logitech C922 Pro USB camera was attached to the follower and used instead. ROS Noetic was used to handle nodes, packages, etc. Both the robots had Raspberry Pi 3B, which are single-board computers (SBC) that can be used to do light computations. The SBCs had SD cards inserted with an Ubuntu image, which allowed the SBC to run the Ubuntu operating system.

The Raspberry Pi SBC has limited RAM and computing power. Therefore, a remote PC was used to communicate with the SBC and do the heavy calculations for it. Heavy calculations on the SBC decreased performance and made the system and sensors connected to it, like the USB camera, lag and publish information at a lower rate than usual. The remote PC was therefore used to run most of the nodes and packages, such as the control nodes, marker detection, etc. The leader and follower Turtlebots had their own remote PCs, and the SBCs and remote PCs were connected through WI-FI and communicated via ROS. The remote PC can launch a ROS master, which handles communication and allows for publishing and subscribing of topics between the robot's SBC and the remote PC. An example of this would be the remote PC being able to publish velocities to the /cmd_vel topic of the SBC, which controls the robot's wheels. The SBC only had to launch a launch file that initialized the Turtlebot's base functionality and basic packages. The remote PC could then launch a launch file that initialized the heavier packages and run the nodes.

In the physical system, similarly to in the simulations, the follower was placed so that the distance between the base_link frame of the follower and the marker was 0.8 meters at initialization. The marker was attached to the back of the leader with double-sided tape.

The initial positions of the two robots in the lab were different depending on the experiment. For example, in experiments where robots drove in a straight line, they started at one side of the room and drove to the other. In other experiments, they started in positions where they could drive in the desired pattern without hitting walls and still be detected by the motion capture system. The world frame was placed in the middle of the lab.

The camera was calibrated, and the re-projection Root Mean Square (RMS) error was around 0.15 pixels. It varied between 0.1 and 0.5 pixels depending on how the checkerboard was held relative to the camera. This means that the calibration was satisfactory, as a re-projection error under 1 pixel is good. The camera's auto-focus was turned off to have a fixed focus. The power line frequency of the camera was edited to fit the lights in the room to make the image have less grain noise.

5.2.1 Physical space and motion capture

In the simulations, there were no walls or other objects aside from the Turtlebots. Because of this, the robots could drive infinitely in any direction. In the real world, space is limited, so some of the experiments had to be changed. The motion capture struggled with detection when the robots were too close to the walls. This was because most of the motion capture cameras focused on the middle of the room, making it so few cameras tracked along the walls. The cameras were moved, adjusted, and the motion capture system was re-calibrated, but detection close to walls was still unreliable.

When doing motion capture, sphere markers were used for tracking. The spheres were placed on the robots, and a minimum of three had to be detected to define a rigid body. A rigid body is required to find the pose. When an almost 20x20 cm ArUco marker was attached to the back of the leader, some of the spherical markers became obscured for cameras in certain positions. This led to the sphere markers not being detected, and with this, a rigid body for the leader could not be defined. Adding more markers made the motion capture more robust. Because of this, more spherical markers were added to the robots, both on the top and on the sides of the robots. If there are five markers and one or two of them are obscured, a rigid body can still be defined. Adding markers to the side of the leader made it so that the ArUco marker did not get in the way of those spherical markers during motion capture.

5.2.2 LiDAR and thresholding

In the simulations, the robots were the only objects in the world. Because of this, the follower's sensors only detected the leader. Therefore, thresholding was not necessary in the simulations. In the real world, there are walls and other objects, and the LiDAR cannot

discern between a wall and the leader. Because of this, the physical system required some form of LiDAR thresholding, as mentioned in section 4.3.1.

In the simulation, if the LiDAR was not detecting anything beyond the max range of the sensor, the range of the laser scan in this position was infinity. Using this, the LiDAR detection in the control node worked by checking if the range of the laser was inf or not to see if a laser ray was detecting anything. In the physical system, the LiDAR returned 0 range instead of inf when nothing was detected.

5.2.3 Physical camera and marker

In the simulations, a Raspberry Pi camera module was used. In the physical system, this camera module was already installed on the Turtlebots. Some time was spent updating packages, changing Ubuntu versions on the SBC, and trying other fixes to make the camera module take pictures or videos, but the camera module did not work. Instead, the camera module was removed and replaced by a USB camera that was taped to the robot in the same position as the camera module. Because the USB camera is not described in the turtlebot_description package, a new frame was defined for the USB camera by publishing a custom frame. This new custom frame replaced the camera_rgb_optical frame that the simulations used. The custom camera frame was the frame of the USB camera image sensor, and the camera measurement data in the physical system were given in relation to this frame. The custom frame was attached to the follower by publishing a custom static transformation between it and the base_link frame.

The USB camera was large compared to the Raspberry Pi Camera module. This became an issue when defining the transformation between the custom camera frame and the base_link frame of the follower. When taking camera measurements, the measurement data is given in the camera frame. This data should be transformed from the camera frame to the base_link frame of the follower. The issue was that the position of the image sensor inside the USB camera was unknown, so it had to be estimated. The height of the camera lens and image sensor from the ground was assumed to be the same. The lens was measured to be 0.105 meters above the ground using a tape measure. Since the base_link is 1 cm over the ground, the image sensor was assumed to be 0.0905 meters over the base_link.

The x-axis of the base_link is pointing in the direction the robot is facing, and the z-axis is pointing up, as shown in Figure 3.3. The z translation of the image sensor relative to the base_link frame was measured to be 0.0905 meters. It was assumed that there was no translation of the camera frame in the y direction of the base_link frame. The x translation of the camera frame relative to the base_link had to be estimated. Because of how big the USB camera was, there could be a potential error of 2 cm in the translation of the x position of the custom camera frame relative to the base_link frame depending on where the sensor was assumed to be inside the camera. Because the camera measurements are transformed

from the camera frame to the base_link frame, the assumption of the position of the sensor in the camera would affect the measurement results.

The translation of the camera frame along the base_link frame's x-axis was estimated by using the LiDAR to find the distance d_1 between the leader and the base_link frame and by using the camera to find the distance d_2 between the camera frame and the marker. The LiDAR data was transformed from the base_scan to base_link, as mentioned in section 4.3.1. Taking $d_1 - d_2$ gives an estimate of the distance between the base_link and the camera frames. Using this, the x translation of the camera frame relative to the base_link was measured to be 4-5 cm. This was not possible. Using a tape measure, the camera was at its closest 7 cm away from the base_link. The reason for the wrong measurements might be that the LiDAR had a bias or that some laser ray measurements were unreliable.

Because the previous estimation of the translation of the image sensor with the LiDAR was unreliable, another method was used. The follower was placed directly behind the leader at a known distance, so the distance between the follower's base_link and the marker was known. A tape measure was used to find this distance. The translation of the custom camera frame along the x-axis of the base_link frame was adjusted until the camera measured the known distance between the base_link and the marker. This gave a translation of 0.079 meters. In addition to the translation, the custom camera frame was rotated to match the orientation of the camera_rgb_optical frame. With this, the full transformation between the custom camera frame and the base_link frame was known.

The marker was placed approximately 15 centimeters above the ground behind the leader. In the simulations, the marker could be perfectly placed at a desired position behind the leader. In the physical setup, the marker might not be placed perfectly vertically or horizontally and might not be exactly 15 cm above the ground.

An issue with using the USB camera instead of the Raspberry Pi was that the USB camera could tilt vertically, and it was hard to know when the camera was perfectly straight. The vertical tilt of the camera was not an issue in the distance-based method; the distance measurements might have been affected, but it was barely noticeable. In the pixel-based method, vertically tilting the camera caused big changes in the measured n-coordinates, which indicated the distance between the vehicles. The tilt had a big effect on the n-coordinate measurements because tilting the camera down a little made the measured pixel point in the image appear considerably further up than if the camera was not tilted. This resulted in higher n-coordinate measurements and vice versa if the camera was tilted up. The Raspberry Pi camera module was held in place by a camera mount and screws. The USB camera was taped to the robot with double-sided tape. The camera being taped, in addition to it being bigger than the Raspberry Pi camera, could lead to the USB camera shaking more when the robot is driving than the Raspberry camera module.

After an experiment, the camera would tilt itself a little up or down because the robots

were shaking while driving. Therefore, it was important to stay consistent with the tilt of the camera throughout the experiments. This was done by placing the follower approximately 0.75 meters behind the leader, and then the transformation between the marker frame and the base_footprint frame was checked. When the camera was approximately straight, it measured the height difference between the follower's base_footprint and marker to be 0.154-0.155 meters. In addition to this, the desired distance and n-coordinates were used. If the camera measured around 187.2 pixels on the n-coordinate and 0.75 meters in the distance when the distance between follower and leader was set to 0.75 meters using a measuring tape, the camera was pointing straight forward. Before every experiment, the camera tilt was checked and set so that the camera was pointing straight forward.

5.2.4 External factors

When going from simulations to the physical setup, some external factors affected the system.

The lighting in the lab was not consistent. Half the room had slightly less light than the other half. In addition, in two opposite sides of the room, close to the walls, the light was stronger than in the rest of the room. The marker detection worked well in the lab, except for in the area with the strong light close to the walls. Here the ArUco marker detection struggled. The detection flickered, meaning that it lost track of the marker often. This did not affect the experiments, as the only time they were under this type of light was in their initial position for the experiments where they were driving in a line. They got out of the strong light quickly in these experiments.

When the Turtlebots drove in the lab, dust and other debris could get stuck to the wheels. This did not seem to affect the experiments. As a precaution, the wheels were cleaned after a few experiments. An issue that could affect the experiments was slow WI-FI. Since the remote PC and SBC were connected through WI-FI, slow WI-FI caused a delay between the nodes being started and the robots reacting. This could also affect the experiments because the information was sent at a slower rate. This was not much of an issue since it was rare for the WI-FI to have so much traffic that it was slow enough to affect the system.

5.2.5 Specifications

Table 5.1 shows the hardware specifications of the Turtlebot3 Waffle Pi. The sensors of the Turtlebot are also listed here.

The LiDAR measurement specifications are given in table 5.2. This table contains information such as the max and min range of the LiDAR, precision, and accuracy.

Turtlebot3 Waffle-Pi hardware specifications	
Items	Waffle Pi
Maximum translational velocity	0.26 m/s
Maximum rotational velocity	1.82 rad/s (104.27 deg/s)
Maximum payload	30kg
Size (L x W x H)	281mm x 306mm x 141mm
Weight (+ SBC + Battery + Sensors)	1.8kg
Threshold of climbing	10 mm or lower
Expected operating time	2h
Expected charging time	2h 30m
SBC (Single Board Computers)	Raspberry Pi
MCU	32-bit ARM Cortex®-M7 with FPU (216 MHz, 462 DMIPS)
Remote Controller	RC-100B + BT-410 Set (Bluetooth 4, BLE)
Actuator	XM430-W210
LDS(Laser Distance Sensor)	360 Laser Distance Sensor LDS-01
Camera	Raspberry Pi Camera Module v2.1
IMU	Gyroscope 3 Axis
	Accelerometer 3 Axis
Power connectors	3.3V / 800mA
	5V / 4A
	12V / 1A
Expansion pins	GPIO 18 pins
	Arduino 32 pin
Peripheral	UART x3, CAN x1, SPI x1, I2C x1, ADC x5, 5pin OLLO x4
DYNAMIXEL ports	RS485 x 3, TTL x 3
Audio	Several programmable beep sequences
Programmable LEDs	User LED x 4
Status LEDs	Board status LED x 1
	Arduino LED x 1
	Power LED x 1
Buttons and Switches	Push buttons x 2, Reset button x 1, Dip switch x 2
Battery	Lithium polymer 11.1V 1800mAh / 19.98Wh 5C
PC connection	USB
Firmware upgrade	via USB / via JTAG
Power adapter (SMPS)	Input : 100-240V, AC 50/60Hz, 1.5A @max
	Output : 12V DC, 5A

Table 5.1: Hardware specifications for the Turtlebot3 Waffle Pi.

The specifications for the Raspberry Pi camera module are given in table 5.3. This module was not used for the physical setup because of issues that made it unusable. This module was, however, used in the simulations

LiDAR measurement specifications	
Items	Specifications
Distance Range	120 $\sim$ 3,500mm
Distance Accuracy (120mm $\sim$ 499mm)	$\pm$15mm
Distance Accuracy(500mm $\sim$ 3,500mm)	$\pm$5.0%
Distance Precision(120mm $\sim$ 499mm)	$\pm$10mm
Distance Precision(500mm $\sim$3,500mm)	$\pm$3.5%
Scan Rate	300$\pm$10 rpm
Angular Range	360deg
Angular Resolution	1deg

Table 5.2: Measurement performance specifications for the LiDAR

Raspberry Pi camera module hardware specifications	
Items	Specifications
Size	Around 25 x 24 x 9 mm
Weight	3g
Still resolution	8 Megapixels
Video modes	1080p30, 720p60 and 640 x 480p60/90
Linux integration	V4L2 driver available
C programming API	OpenMAX IL and others available
Sensor	Sony IMX219
Sensor	3280 x 2464 pixels
Sensor	3.68 x 2.76 mm (4.6 mm diagonal)
Pixel size	1.12 μm x 1.12 μm
Optical size	1/4”
Full-frame SLR lens equivalent	35 mm
S/N ratio	36 dB
Dynamic range	67 dB @ 8x gain
Sensitivity	680 mV/lux-sec
Dark current	16 mV/sec @ 60 C
Well capacity	4.3 Ke-
Fixed focus	1 m to infinity
Focal length	3.04 mm
Horizontal field of view	62.2 degrees
Vertical field of view	48.8 degrees
Focal ratio (F-Stop)	2.0

Table 5.3: Specification for the Raspberry Pi v2.1 camera module

Table 5.4 shows the specifications of the Logitech C922 Pro USB camera that was used in the physical setup.

Logitech C922 Pro specifications	
Items	Specifications
Height	44 mm
Width	95 mm
Depth	71 mm
Weight	162 g
Max Resolution	1080p/30fps - 720p/60fps
Camera Megapixel:	3
Diagonal Field of View (dFOV)	78°

Table 5.4: Specification for the C922 Pro Logitech USB camera

5.2.6 Technical issues/ Bugs

There were some technical issues encountered in the physical system and simulations. The first issue was using two Turtlebots at the same time in the simulations. At the time when the simulations were being set up, the TF prefix parameter in ROS Noetic was not supported. The TF tree of the robots would not be defined correctly, and ROS would have issues differentiating between the frames of the two robots because there were no prefixes for the robots or frames. This made it so that there was, for example, only one base_link frame. But since both robot frames were published without prefixes, the single base_link frame would switch between being the follower or leader's frame constantly. To solve this, the Turtlebot description files had to be edited so that they allowed prefixes to be set. An example of the prefixes can be seen in figure 3.2, where the frames of the leader have the $tb3\_0$ prefix.

Another technical issue was with the aruco_detect package. In this package, there is a parameter called $maxMarkerPerimeterRate$, which determines the maximum perimeter rate for marker contours to be detected. According to the documentation, the default value for this parameter is 4. There is a bug that does not let this parameter go over or be set to anything over 1, so the parameter is set to 1 by default. Having a value of 1 on this parameter made the marker detection stop detecting the marker if it got too close to the camera, despite the marker still being in perfect view of the camera. To fix this bug, one of the files in the aruco_detection package had to be edited so that the upper boundary for the parameter was 4 instead of 1.

This bug affected the pixel-based method greatly. In this thesis, the FoV constraints for the pixel-based method were found using the camera. The bug would make the measured n_{min} be 170-180 pixels because the camera would stop detecting the marker if the camera got too close. In the simulations, this issue affected the performance of the pixel-based method, but the method was still stable. In the physical setup, however, this bug made it so the system could not be kept stable for more than 30-40 seconds at max. After the bug

was resolved, n_{min} would be measured to be around 140-145 pixels. The lower n_{min} made the performance of the pixel-based method in the simulations considerably better, and the method could be kept stable for the physical experiments.

In the physical system, there was an issue with the LiDAR that was not resolved and persisted in the experiments. The LiDAR measured distances shorter (2-3 cm) than it should. This was first thought to be noisy measurements pulling the measured distance lower. An additional set of thresholds were tested to remove more noise. A threshold for the minimum distance was set so that the LiDAR detection area became the edge of a cone instead of a cone. This did not solve the issue. An attempt at setting the thresholds very small was also made, but this removed legitimate measurements while still giving smaller distance measurements.

When placing the follower 0.75 meters behind the leader, the LiDAR measured a distance of a little under 0.73 meters. The transformation from the base_scan to base_link was checked in case this was the reason for the lower distance. When using the non-transformed LiDAR data and placing the follower 0.75 meters behind the follower, the distance in the base_scan should measure $0.75 + 0.064 = 0.814$ meters, since 0.064 meters is the translation between the two frames. The distance measured was closer to 0.79 meters, showing that there might be a bias in the LiDAR sensor.

5.3 Software Setup

Most of the code in this thesis was written in Python, and some minor nodes were also written in C++. The main nodes will be presented in this section. The explanations will assume that all parameters and constants for control have been set.

3 custom nodes were used to run the experiments in this thesis. A leader control node, control nodes for the follower, and a position tracking node. The leader control node published values to the leader's /cmd_vel topic. There were four different experiments, with four different patterns that the leader could drive in. There was a parameter in the leader control node that could be set between 1 and 4 depending on the pattern that the leader should drive in. This was set up with if statements. For example, if the parameter was set to 1, the linear and angular velocities that were required for the leader to drive in circle patterns were published to the leader's /cmd_vel topic. The control of the robot was inside a while loop that published velocities to the /cmd_vel topic at a certain rate, where the published values were continuously updated. The 3 custom nodes were set to run at 10 Hz.

The second node was the control node of the follower. The node used here depends on which controller and version (camera-only, LiDAR-only, etc.) of the controller should be used. Because of this, there were 8 different control nodes for the follower. Four for the

distance-based method and four for the pixel-based method. For the camera-only version of the distance-based method, the node first checked if the marker was detected. If the marker was detected, the node could subscribe to the /fiducial_transforms topic. The topic gave the pose of the marker in the camera frame. The pose was transformed to the follower's base_link frame and used to find the distance and bearing between the base_link frame and the marker, which were sent to a control function. The control function calculated the linear and angular velocities based on equation (8). The velocities were published to the /cmd_vel topic of the follower. This was in a loop so that the velocities published to the topic were continuously updated based on the measurements. The additions of the other versions of the method were added to the nodes of the respective versions. For the LiDAR-only version, the /scan topic was subscribed to, which gave raw information about the LiDAR measurements. There was a callback function that received information from the /scan topic and sent the distance and bearing measurements to the control function after applying the transforms and thresholds, as shown in sections 4.3.1 and 4.3.2. In the filtered version, the two previous velocities were stored so they could be used for filtering, as mentioned in section 4.2. For the complete version, the camera-only and LiDAR-only version nodes were merged and measurements from the LiDAR and camera were combined, as shown in section 4.4.1. The previous velocities were stored to apply filtering.

For the pixel-based method, the control nodes were set up similarly to the distance-based method. Information from sensors was sent to the control function, velocities were calculated, and the control function published velocities to the /cmd_vel topic, and this process was in a loop. The difference in the pixel-based method was that in the camera-only version, instead of sending the distance and bearing measurements to the control function, the pixel coordinates of the center of the marker were sent. This was done by subscribing to the /fiducial_vertices topic, which was published by the aruco_detect package. This topic published the pixel coordinates of the corners of the marker in the image, which were used to find the marker center. After the pixel coordinates were sent to the control function, the velocities were calculated based on equation (18). The other nodes for the different versions of the controller were implemented in a similar way to how it was done for the distance-based method. There was a difference in the versions that used the LiDAR because the pixel coordinates had to be estimated, as shown in section 4.3.3. A race condition occurred if a node tried to use information from a subscribed topic with no messages, because of this, the nodes had to wait for the first message to appear on the topic.

The final node was a node that tracked the position of the two Turtlebots relative to the world frame by using the TF2 package. For the physical experiments, only the nodes controlling the leader and follower were used. This is because the motion capture system was used to track the positions of the Turtlebots, so the final node was replaced by the motion capture system in the physical setup.

An example of how the different nodes and topics were connected is shown in figure 5.1. This figure is for the simulated complete version of the distance-based method. The custom

nodes are the /simple_follow, /decentralized_complete, and /position_plotter, which are the leader, follower, and position nodes respectively. In the figure, it can be seen how some of the different topics, nodes, and packages mentioned in 3.1.8 and 3.1.9 are connected to the custom nodes. For example, how the /decentralized_complete node is subscribing to the /tf topic.

As seen in section 5.2.5, the Turtlebot Waffle Pis had maximum/minimum velocities. If the Turtlebot's velocity exceeds the maximum velocity, it should automatically be set to the maximum velocity. In the simulations, this did not happen. Therefore, velocity boundaries were set in the control nodes so that the velocity of the Turtlebots would not exceed maximum velocity. In the physical setup, the Turtlebots cannot physically drive faster than the maximum/minimum velocities.

Timing in the nodes was done by taking the difference between the node's current time and the start time of the node. In the physical system, these times followed the wall-clock time. In the simulations, the times were kept track of by a simulated clock in ROS. Race conditions could occur that set the start time of the node to 0. This occurred if the timing of the node was initialized before the /clock topic. To avoid this, the node waited until the start time of the node was a non-zero value. Handling time properly was important in all 3 of the custom nodes presented in this section, both for control and to get accurate time-stamped data.

5.3.1 Data collection

It is important to be able to store data to get tables and plots of the data. The nodes could handle plotting directly. They did this by appending data values from every iteration of the control loop into lists for that specific parameter and then plotting the lists. For example, lists for the linear or angular velocities. An alternative to storing data in lists would be to use Rosbag files, which are files that record the messages in the topics of the nodes. The data that is going to be in the tables or be plotted was not published to topics, so it was more effective to use lists. The leader control node stored the time and the leader's linear and angular velocity. The follower control nodes stored the time, the follower's linear and angular velocity, the errors and error boundaries, and the relative distance and bearing angle between the follower's base_link and marker frame. The boundaries and errors that were stored depend on the method. For the distance-based method, the heading and distance errors with the boundaries were stored. For the pixel-based method, the pixel errors and the pixel error boundaries were stored. The position node stored the time and the leader and follower's positions.

After storing all the data in lists, these lists can either be plotted directly or be written into Comma-Separated Values (CSV) files. The advantage of storing the lists in CSV files is that the data is available at any time without having to run the experiments again, so the

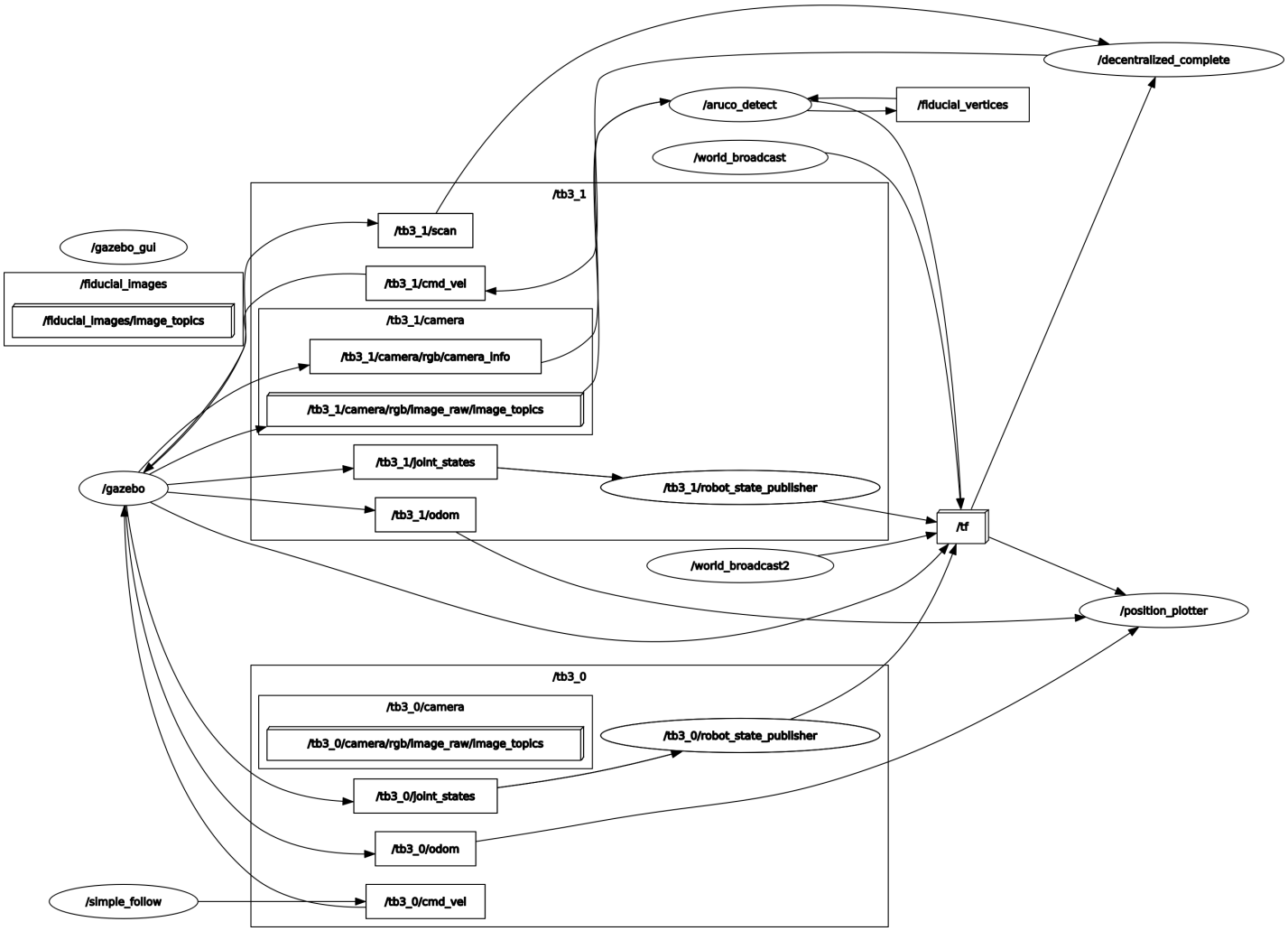

Figure 5.1: A figure of the different nodes and topics when running the simulated complete version of the distance-based method. The circles are nodes, and the squares are topics. The arrows in and out of the topics/nodes show which topics/nodes they are subscribing and publishing to. The custom nodes are /simple_follow, /position_plotter, and /decentralized_complete. /tb3_0 and /tb3_1 are the prefixes of the leader and follower respectively.

data stored in the files can be used later for plotting and making tables. It is easier to work with the data and plot graphs of several runs in one plot when using CSV files. The new data values were stored in the list every time the control loop/function was iterated.

In the simulations, the position node was used to store position data. In the physical setup, the motion capture system was used to get the position data instead. The position data in the physical setup was stored directly in Tab-Separated Values (TSV) files, which are similar to CSV files. Using the position data, the RMS of the position error can be found. The data from the motion capture system is almost identical to the data that would have been obtained by the position node.

In the pixel-based method, the distance and bearing were measured during the experiments but not used in the controller; they were only measured to compare the controllers. After obtaining these measurements, the distance and heading errors in the pixel-based method were found by subtracting the desired distance and desired bearing angle from the measurements. The standard deviation was taken directly from the measurements. The sensors used to gather the distance, bearing, and image coordinates data depend on the sensors used in the experiment. In the camera-only and filtered versions of the controllers, the camera was used to obtain the measurements. In the LiDAR-based method, the LiDAR was used. In the complete version of the distance-based method, the controller used combined measurements from the LiDAR and camera to measure distance and bearing. For the pixel-based complete version, the LiDAR and camera measurements were combined when measuring the pixel coordinates, but when the distance and bearings were measured, only the camera was used.

5.4 Experiments

The experiments consisted of making the leader drive in some pattern while the follower was detecting the leader and following it based on measurements from sensors. The controller took the measurements and calculated velocities to control the follower. The robots were driving in 4 different, separate patterns. The different patterns were: the robots drive in a circle. The robots drive in a straight line with constant velocity. The robots drive in a figure-8 once. The robots drive in a straight line with a velocity that gradually accelerates up to a certain velocity before slowing down gradually. Table 5.5 shows the different experiments, controllers, and versions of the controllers.

In the simulations, for the experiment where the robots drove in a circle, the leader had a constant angular velocity of 0.1 m/s and a constant linear velocity of 0.2 m/s. This experiment lasted 300 seconds. For the experiment where the robots drove in a straight line with constant velocity, the leader had a constant linear velocity of 0.2 m/s and an angular velocity of 0 m/s. This experiment lasted 200 seconds. In the experiment where the robots drove in a figure-8, the leader started by driving with a constant linear velocity of 0.2 m/s

Methods	Versions	Patterns
Distance-based method	Camera-only	Circle
Pixel-based method	LiDAR-only	Line with constant velocity
	Filtered	Figure-8
	Complete	Line with dynamic velocity

Table 5.5: Table of methods, the version of these methods, and the different patterns in the experiments.

and angular velocity of 0.1 m/s. When the leader had completed one circle, which took 64 seconds, the angular velocity changed to -0.1 m/s. After 128 seconds, the robots had driven in a figure-8 pattern. In the experiment where the robots drove in a straight line with dynamic velocity, the leader started with a velocity of 0.1 m/s. It drove with this velocity for 40 seconds before accelerating to 0.15 m/s. After this, every 20 seconds, it added another 0.05 m/s until it reached a velocity of 0.25 m/s. Once this velocity was reached, the leader started driving 0.05 m/s slower every 20 seconds until it finally stopped completely. This experiment took 160 seconds in total.

The experiments in the physical setup had to be adjusted. The robots should not touch the walls, and since a motion capture system was used, the experiments were adjusted so that the robots were always in view of enough cameras for robust motion tracking. Some areas closer to the walls had unreliable motion tracking because not enough cameras could track the robots there. The circle experiment remained the same as in the simulations. In the experiment where the robots drove in a line with constant velocities, the velocities remained the same as in the simulations. The difference is that this experiment only lasted for 43 seconds here. For the figure-8 experiment, the leader drove with a linear velocity of 0.2 m/s and angular velocity of -0.1 m/s for 66 seconds, then switched to an angular velocity of 0.1 m/s. The leader drove like this for another 66 seconds, creating a figure 8 after 132 seconds. In the experiment where the robots drove in a line with dynamic velocity, the leader had no angular velocity, and the linear velocity started at 0.1 m/s. The leader drove with this velocity for 30 seconds, increasing to 0.15 m/s and 0.2 m/s at 30 and 40 seconds respectively. After this, the velocity decreased to 0.15 m/s at 50 seconds and 0.1 m/s at 60 seconds. This experiment lasted for 70 seconds.

Table 5.6 shows the different velocities in the experiments and the duration of the experiments. Figure 5.2 shows the velocity profiles for the experiment where the robots drove with dynamic velocity, both for the simulations and physical setup. As previously shown, there were four different versions of the two methods. These four versions were tested in the four experiments.

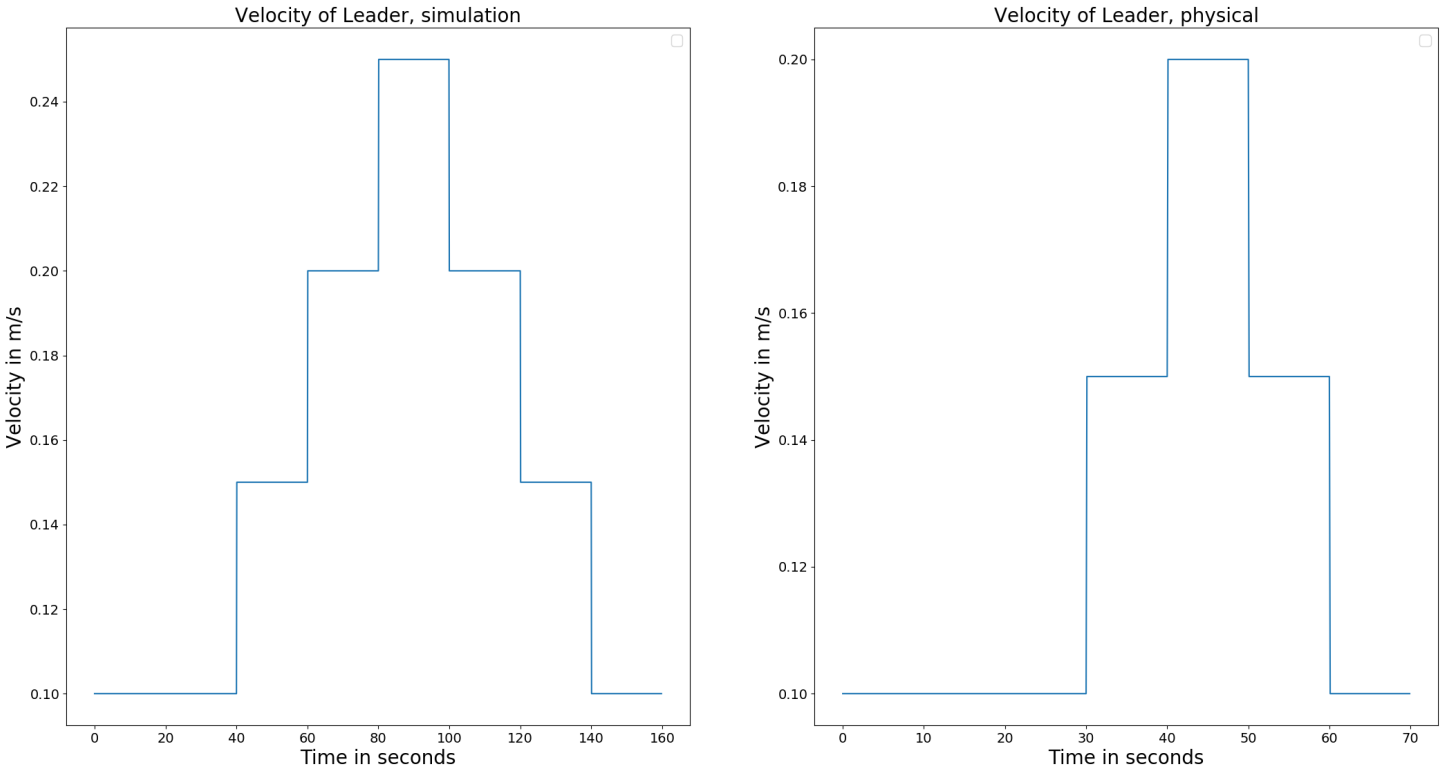

Figure 5.2: A figure of the leader's linear velocity profile in the experiment where the robots drove with dynamic velocity. The plot on the left shows the velocity profile for the simulations, and the right plot shows the velocity profile for the physical setup.

Table of the different experiments.			
Experiments	Linear Velocity(m/s)	Angular Velocity(rad/s)	Duration(s)
Circle	0.2	0.1	300
Line with constant velocity	0.2	0	200 **(43)**
Figure-8	0.2	0.1→-0.1 **(-0.1→0.1)**	128 **(132)**
Line with dynamic velocity	0.1→0.25→0.1 **(0.1→0.2→0.1)**	0	160 **(70)**

Table 5.6: The table shows the experiments, the linear and angular velocities of the leader, and the duration of the experiments, both for the simulations and physical experiments. The bold numbers in the parentheses are for the physical experiment if there were changes between the physical experiments and simulations. The → indicates a change in velocity from the minimum velocity in the experiment to the max velocity.

5.4.1 Performance metrics

The results will be shown in the form of tables with mean and standard deviation. In the experiment where the robots drove in a figure-8 and a line with dynamic velocity, the velocities changed during the experiment. Because of this, the standard deviation was not only affected by noise but also by how good the system was at adjusting to new velocities. For these experiments, the mean is a bias or tracking error, while the standard deviation is a variation around this.

When calculating the mean and standard deviation, the first 35 seconds of the runs were removed. This is because at the start of the experiment there is a transient which should not be included to get a more accurate sense of the mean and standard deviation. At 35 seconds, the transient was not completely gone, but taking more than this was not viable because the physical experiment where the robots drove with constant velocity only lasted for 43 seconds.

In most experiments, the RMS of the position error of the two robots was calculated. When calculating the RMS of the position error, the leader was shifted back to the time instance when its position was approximately in the same position as the follower. This was done by adding a time delay to the leader's position data to shift it back in time. Then the position error between the two robots was taken. It was assumed that the linear velocity of the two robots was constant so that a delay to the leader's position data would shift it to the position of the follower. Because of this, the RMS of the position error was not taken for the experiment where the robots drove in a line with dynamic linear velocity.

The RMS of the position error is an estimate, and how good the estimate is depends on

the accuracy of the time delay. The delay was found through testing. This was done by testing different time delays and looking at the positions of the two robots so that they lined up as much as possible when the leader's position was shifted back. The delay was different depending on if the pixel or distance-based method was used, and it also depended on the relative distance between robots throughout the experiment.

When evaluating the experiments, different metrics are used to compare the results. The controller is given measurements and calculates a measurement error and velocities based on this in an attempt to minimize the measurement error. In the distance-based method, the controller tried to minimize the distance and heading errors shown in equation (3). In the pixel-based method, the controller tried to minimize the pixel coordinate errors shown in equation (13). The linear and angular velocity error means shows the difference between the leader and follower's velocities, and how well the follower measures the leader's velocities and follows them. The noise/standard deviation of the data is important as it indicates the stability of the system. The position error shows how well the follower is following the leader's trajectory. High position error means that the follower is not following the trajectory well. The standard deviation of the position error will not be looked at closely. The measurement and velocity errors, standard deviations of these errors, and position errors will be looked at when comparing the versions and methods. In the pixel-based method, distance and heading errors were also taken to have more metrics to compare the two controllers.

5.5 Choice of parameter values

In this section, the choice of the values for parameters will be looked at. In the controllers there were parameters such as the control gain parameters, k_d and k_β for the distance-based method, and k_n and k_m for the pixel-based method. Parameter values had to be chosen and tuned for the controllers and the experiments. This had to be done for both controllers. The main experiments in this thesis will use the parameters mentioned in this section. There were also some experiments done to tune parameters or to show the effects of different parameter values on the system. These will use most of the same parameter values as the main experiments but will use different values on some of the parameters.

5.5.1 Choice of parameter values for simulations

For the distance-based method, the maximum steady-state errors were set to $\rho_{d,\infty} = 0.2$ and $\rho_{\beta,\infty} = 8$. β_{con} was set to $\frac{AoV}{2}$, where angle of view (AoV) is the horizontal FoV of the camera. AoV was found to be 60 degrees. The desired distance between the leader and follower was $d_{des} = 0.75$ meters, and the desired bearing angle was 0 degrees. The collision distance was set to $d_{col} = 0.05 d_{des}$ meters, and the distance where the camera stopped detecting the

marker was $d_{con} = 3.15$ meters. l, which is the convergence rate, was set to 0.1.

For the pixel-based method, the maximum steady-state errors were set to $\rho_{n,\infty} = 20$ and $\rho_{m,\infty} = 30$. h, which was the height difference between the camera optical axis and the marker center, was set to -0.0493 meters. h could be found through the transformation between the marker and camera frames or measured directly. It could also be found by placing the follower in a known position behind the leader, measuring the n-coordinate, and using this to calculate h through equations (10) and (11). The desired position of the follower was 0.75 meters directly behind the leader, but in this method, it had to be converted into pixel coordinates. This was done by inserting the desired position of the leader in the follower frame, (0.75,0), into equation (10) so that (X,Y,Z) = $(-y_{lf}, h, x_{lf})$ = (0,-0.0493,0.75). Inserting this into equation (11) gave approximately $m_{des} = 320$ and $n_{des} = 199.5$ pixels. An alternative way to find the desired pixel coordinates was to place the follower in the desired position behind the leader and measure the desired pixel coordinates directly with the aruco_detect package.

m_0 and n_0 were the pixel coordinates of the projective center of the camera and could be found in the camera calibration matrix. In the simulations, the camera calibration matrix was ideal. When using a 640x480 resolution with an ideal camera, $m_0 = 320$ and $n_0 = 240$ pixels, meaning exactly in the middle of the camera. α_n and α_m were set to 616. It was not mentioned how α_n and α_m were selected in (Miao et al., 2021), just that they were constant scaling factors. In this thesis, it was assumed that α_n and α_m were the focal lengths expressed in pixel units and could be found in the camera calibration matrix. This was assumed because equation (11) came from the pinhole camera model. The convergence rate l was set to 0.1.

The FoV constraints of the pixel-based method had to be set. In (Miao et al., 2021), they used a resolution of 640x480. It was not mentioned explicitly what the FoV constraints in the paper were, but they were assumed to be $m_{max} = 640$, $m_{min} = 0$, $n_{max} = 480$, $n_{min} = 0$ pixels. These FoV constraints were tested but made the follower drive a couple of meters backward before very slowly approaching the desired position behind the leader. This was not ideal, as the follower could easily lose track of the leader if the leader was driving while the follower was backing up initially. The follower did not back away initially in (Miao et al., 2021) with the static gain controller, and it could be because of how their setup differed from the one in this thesis. Their camera and tracking object were on poles of different heights, which gave a larger height difference between the camera's optical axis and the tracking object.

New FoV constraints were tested and used. They were found by letting the follower stand still while detecting the pixel positions of the leader's marker center. The leader was moved around on the ground while in view of the camera. The leader was moved as close, as far away, and as far to the sides of the follower as possible while still being in the camera's FoV. The constraints were the pixel coordinates where the camera stopped detecting the marker if

the marker went out of these coordinates. These were: $m_{max} = 620$, $m_{min} = 20$, $n_{max} = 235$, $n_{min} = 145$ pixels. With these FoV constraints, the follower immediately followed the leader without backing up initially. For the distance-based method, the AoV and d_{con} were found in a similar way by using the distance and bearing measurements instead.

5.5.2 Choice of parameter values for physical setup

For the physical distance-based method, the maximum steady-state errors were set to $\rho_{d,\infty} = 0.2$ and $\rho_{\beta,\infty} = 12$. The distance where the camera stopped detecting the marker was $d_{con} = 3.1$. Aside from that, the same values as in the simulated distance-based method in section 5.5.1 were used.

In the pixel-based method, the maximum steady-state errors were set to $\rho_{n,\infty} = 30$ and $\rho_{m,\infty} = 60$. The center pixels $n0$ and $m0$, as well as the scaling factors α_n and α_m were taken from the camera calibration matrix. These were $n0 = 235.6$ pixels, $m0 = 315.26$ pixels, $\alpha_n = 651.61$, and $\alpha_m = 649.018$. Other values were given as $h = -0.0577$ meters, $n_{des} = 187.2$ pixels and $m_{des} = 315.26$ pixels. The FoV constraints were $m_{max} = 620$, $m_{min} = 20$, $n_{max} = 230$, $n_{min} = 140$ pixels. n_{des}, m_{des}, h and the FoV constraints were found in the same way as in section 5.5.1. Aside from that, the same values as in the simulated pixel-based method in section 5.5.1 were used.

5.5.3 Choice of control loop frequency

The rate of the control loop had to be chosen. As mentioned, topics can be subscribed or published to. The rate of the control loop decides the frequency at which the control equations are calculated and velocities are published to the robot wheels. The rate of the /scan topic was around 6-7 Hz. The publishing rate of the camera depends on its Frames Per Second (FPS), this is usually 30 or 60 Hz but depends on the resolution and camera. Since the physical Turtlebot's SBC lacked computing power, the USB camera got less than 30 FPS. The FPS was around 20 with black and white video while the image processing was active to rectify the image with the camera calibration. The image processing was done on the SBC because it could not be done through the remote PC due to lag. Black and white video was used in the physical system because this gave a little more FPS than colored video. There was no lack of computing power in the simulations, so the camera had the full FPS.

Generally, choosing a high frequency for the control loop is good if the system has enough computing power and a high rate of measurements. A high rate on the control loop can lead to unnecessary noise, which must be filtered out. In this thesis, a lower frequency rate was chosen. The physical Turtlebot would struggle with computing the control equations at 100 Hz, for example, since it has limited computing power. Even when using a remote PC to

compensate for the computing power, publishing 100 velocities to the wheels every second would be excessive. Another reason for not having a high control loop rate was that the measurements from the sensors were updated around 5-20 times a second. If the control loop ran 100 times a second, many of the control loop velocity computations would be done with the same measurements. There would be no point in computing the velocities this often.

Having a too low rate on the control loop can cause issues. If the rate is too low and the leader is moving fast, the follower could struggle to keep up with the leader when it comes to publishing new velocities to the wheels. Another issue is if the control loop is so slow that it misses many measurement updates. A rate of 135 Hz was tested in the simulations. This gave more noise than usual, especially in the versions of the methods that were subscribing to the /scan topic, which gave information about the LiDAR measurements. This was because the LiDAR published at a low frequency. Having a rate this high was excessive, and it was too high for the physical system. A rate of 5 Hz was tested for the control loop, but in some cases, the system could not keep up with the leader with this rate. In the end, the loop rate was set to 10 Hz. With this, the system managed to keep up with the leader while not introducing too much noise.

5.5.4 Optimization of error boundaries and control gain

For the system to remain stable, the measurement error must not go outside the error boundary. As such, the boundaries were chosen so that even if there were noise and error spikes in the system, the error should not go outside the boundary set by the prescribed performance. If the error went outside of the boundary, the follower would not be able to follow the leader. This was because the system became unstable since values in the controller became NaN or infinite. While keeping the boundaries large held the system stable in cases with error spikes and noise, the boundaries should be pushed as low as possible. This is because lower boundaries give better performance in the controller. It was possible to optimize so that the error boundaries were as small as possible while still keeping the system stable.

The prescribed performance boundaries are affected by the maximum steady-state errors. These are $\rho_{d,\infty}$ and $\rho_{\beta,\infty}$ for the distance-based method. The boundary equation is given in (4). In addition to the steady-state error, the convergence rate l and the constraints given in equations (6) also affect the boundaries, as seen in equation (5). The steady-state errors are $\rho_{n,\infty}$ and $\rho_{m,\infty}$ for the pixel-based method, with the boundary given in (14). Here the convergence rate l also affects the boundaries. In addition, the constraints given in (16) affect the boundaries, as seen in equation (15).

An attempt at optimizing the boundaries and control gain values was made. For the optimization, the circle experiment was used. This was because it was the most demanding experiment, as it ran for 300 seconds while using both linear and angular velocity. When

optimizing the two control methods, the LiDAR or camera-only versions of the controllers were used depending on which was the most unstable and noisy. This was done under the assumption that one of these versions was the most unstable and noisy since they were the simplest versions without any additions, unlike the filtered and complete versions. This was also done under the assumption that if the most unstable and noisy version of the method could handle the optimized parameters, the other versions of the method could also be kept stable under the same parameters. For the simulations, the LiDAR-only versions were used to optimize both control methods. The LiDAR-only version was also used for the optimization of the physical distance-based method. The camera-only version was used to optimize the physical pixel-based method, as this was the least stable and most noisy version of the method.

When optimizing the boundaries, the maximum steady-state errors were tuned to make the boundaries as small as possible while still keeping the system stable. First, large values of steady-state errors were tested in the circle experiment with different control gain values. If these managed to keep the system stable for the entire duration of the experiment, the steady-state errors were lowered a little. After adjusting the steady-state errors, they were tested with different combinations of control gain values. If system instability occurred, the steady-state error had to be readjusted by, for example, raising one of the steady-state errors a little. This tuning process was repeated until the system could not be kept stable for any tested control gain. After optimizing the steady-state errors with the circle experiment, control gains were tested for the other experiments.

In the simulations, the maximum steady-state errors could be set quite low. Starting at $\rho_{d,\infty} = 0.4$, $\rho_{\beta,\infty} = 15$ for the distance-based method and $\rho_{n,\infty} = 30$, $\rho_{m,\infty} = 60$ for the pixel-based method and ending at $\rho_{d,\infty} = 0.2$, $\rho_{\beta,\infty} = 8$ and $\rho_{n,\infty} = 20$, $\rho_{m,\infty} = 30$. This was because of the low noise levels in the simulations, which allowed the steady-state errors to be pushed down considerably.

In the physical system, the maximum steady-state errors started with the same values as in the simulations, and ended at $\rho_{d,\infty} = 0.2$, $\rho_{\beta,\infty} = 12$ and $\rho_{n,\infty} = 30$, $\rho_{m,\infty} = 60$. The distance-based method's steady-state errors could be lowered, but not as much as in the simulations. The physical pixel-based method did not handle a lowering of the steady-state error at all. This was because the pixel-based method had a large amount of noise and error spikes in its camera-only version. This made the system unstable if the boundaries were lowered since the error became bigger than the boundary values.

When having the boundaries optimized, there were only a small number of control gain combinations that could keep the system stable. As opposed to when having a larger boundary, where a wider variety of control gain combinations could keep the system stable. An example of this is the distance-based method in the simulations. With the maximum steady-state errors $\rho_{d,\infty} = 0.4$ and $\rho_{\beta,\infty} = 15$, the control gains could be chosen between $k_d = 0.3{-}0.5$ and $k_\beta = 0.5 - 1$. Most combinations of these would keep the system stable and could even

Simulated distance-based method, robots driving in a circle, different $\rho_{d,\infty}$ values.		
$\rho_{d,\infty}$ value	0.3	0.4
Velocity(m/s)	0.1906± 0.02015	0.18965± 0.01482
Angular Velocity(rad/s)	0.09937± 0.01526	0.0998± 0.0105
Velocity Error Mean (m/s)	-0.0094± 0.02015	-0.01035± 0.01482
Angular Velocity Error Mean (rad/s)	-0.00063± 0.01526	-0.0002± 0.0105
Distance Error(m)	0.05112± 0.00622	0.0676± 0.00607
Heading Error(deg.)	4.14106± 0.59846	4.16899± 0.41416
RMS of position error(m)	0.07144± 0.00238	0.07968± 0.00194

Table 5.7: Table of runs of the simulated distance-based method, LiDAR version. The robots drove in circles. Two different $\rho_{d,\infty}$ values were tested. The table consists of the mean± standard deviation.

keep the system stable in all of the four different experiments. When lowering the steady-state errors to $\rho_{d,\infty} = 0.2$ and $\rho_{\beta,\infty} = 8$, the combinations and range of variety of control gains that could keep the system stable in the different experiments became lower. What control gains can keep the system stable depend on the steady-state errors.

Table 5.7 shows the results of an experiment where the robots drove in a circle for 300 seconds with different error boundaries. The LiDAR version of the simulated distance-based method was used with two different values for the $\rho_{d,\infty}$ to show how different boundaries affected the system. Only $\rho_{d,\infty}$ was changed so that the effects of the change were clearer. Changing this value changes the boundary for the distance error in the system. $\rho_{\beta,\infty} = 10$, $k_d = 0.3$, and $k_\beta = 0.7$ here. The rest of the parameter values for this experiment were chosen as described earlier for the simulated distance-based method.

5.5.5 Choice of control gain

The control gain values are important as they decide how much the system should react to errors. These parameters are a form of scaling factor for the control velocity equations, as can be seen in equations (8) and (18). Large control gain values can result in high velocities if the error is big, and it also makes the system react more to small errors because the control gain parameters scale the error up. This can result in the control velocity overshooting the target

velocity by a large amount, which could lead to collisions. Another issue with having high control gain is that it can cause oscillations. A little oscillation is expected. The ideal scenario is that the system overshoots a little and then settles to the target velocity with minor to no oscillations. In cases where the control gain parameters are too big, the oscillations can become amplified after the system first overshoots. The system keeps undershooting and overshooting as the controller is attempting to minimize the measurement error, amplifying the oscillations until the system becomes unstable. A system with small gain values reacts less to errors, but too small gain values can make the system have issues with following the leader. This is because the gains are so small that the system does not react appropriately until the error becomes big, and at that point, it could be that the follower has lost track of the leader.

The different experiments required different control gains based on which experiment was done. The control gains were chosen based on the trajectory of the leader. If the leader is driving in a straight line with a high velocity, the control gain for the linear velocity controller should be high, while the gain for the angular velocity controller should be low. This is because the leader has no angular velocity, and having a high value for the angular velocity gain could make the system have oscillations in the angular velocity, and in the worst case, become unstable. The controller requires a high value in the linear velocity controller to keep up with the leader. The same control gain values should not be used in an experiment where the leader is driving in a circle since that requires high angular velocity. The follower would not be able to keep up with the angular velocity of the leader. If the leader is driving slowly in a circle, the large linear velocity gain could also cause oscillations or instability in the linear velocity. Therefore, the control gains were chosen according to the pattern in the experiment. An idea for getting the right gains could be to have a form of adaptive parameter tuning. In this thesis, the control gains were found through trial and error.

To show how different control gain values affected the system, an experiment was done on the distance-based method, with the camera-only version. In the experiment, the robots drove in a circle with different control gains in the simulations for 300 seconds. The control gain pairs $(k_d, k_\beta) = (0.3, 0.5), (0.3, 0.7), (0.5, 1)$ were used. Here the maximum steady-state errors were $\rho_{d,\infty} = 0.4$ and $\rho_{\beta,\infty} = 10$. The rest of the parameter values for this experiment were chosen as described earlier for the simulated distance-based method. Table 5.8 shows the results of the experiments where different control gains were used. Only one pair of control gains will be used when showing the results of the other experiments. This means that for every experiment that is done, only one pair of control gains will be used for the method and the particular experiment.

The control gain values that were chosen for the experiments with the simulated distance-based method were $(k_d, k_\beta) = (0.2, 0.5)$ for the circle experiment, (0.25,0.1) for the experiments where the robots drove in a line with constant velocity, (0.2,0.5) for the experiment where they drove in a figure-8, and (0.2,0.15) for the experiment where they drove in a line

Distance-based method, camera-only, circle experiment. Runs with different control gains			
Control gains	0.3,0.5	0.3,0.7	0.5,1
Velocity (m/s)	0.19157± 0.00072	0.18957± 0.00094	0.18921± 0.01404
Angular Velocity (rad/s)	0.09999± 0.00053	0.1± 0.00056	0.09913± 0.00544
Velocity Error Mean (m/s)	-0.00843± 0.00072	-0.01043± 0.00094	-0.01079± 0.01404
Angular Velocity Error Mean (rad/s)	-1e-05± 0.00053	-0.0± 0.00056	-0.00087± 0.00544
Distance Error (m)	0.06764± 0.00122	0.06684± 0.00122	0.03792± 0.00311
Heading Error (deg.)	5.55776± 0.081	4.18741± 0.06857	2.91605± 0.17583
RMS of position error (m)	0.06004± 0.00063	0.07422± 0.00051	0.08223± 0.00112

Table 5.8: Table of runs of the simulated distance-based method, camera-only version. Different control gains were used. The robots were driving in circles. The table consists of the mean± standard deviation.

with dynamic velocity. For the simulated pixel-based method these were $(k_n, k_m) = (0.4, 0.1)$, (0.4,0.02), (0.4,0.1) and (0.45,0.04) respectively. For the physical distance-based method, they were chosen as $(k_d, k_\beta) = (0.2, 0.6)$ for the circle experiment, (0.2,0.1) for the experiment where the robots drove in a line with constant velocity, (0.18,0.6) for the experiment where they drove in a figure-8, and (0.17,0.1) for the experiment where they drove in a line with dynamic velocity. For the physical pixel-based method the control gains were $(k_n, k_m) = (0.4, 0.08)$, (0.4,0.02), (0.4,0.08), and (0.38,0.02) respectively. These were the control gains that were used for the main experiments, meaning the four experiments with the four versions of the methods.

5.5.6 Choice of weights in the filter and the complete version

In section 4.2, it was mentioned that a 2nd-degree filter would be used. This filter has 3 weights, a, b and c. Initially, 1st degree filters with the weights $a = 0.3$, $b = 0.7$ and $a = 0.5$, $b = 0.5$ were tested, but this did not filter out noise very well. The 1st-degree filter was then switched out with a 2nd-degree filter, which removed more noise. Different weights were tested for the 2nd-degree filter. Taking the average across the three velocities gave a good amount of filtering. Because of this, $a = b = c = \frac{1}{3}$

When using the complete version of the methods, the weights w_1 and w_2 of the LiDAR and

Simulated distance-based method, circle experiment. Runs with different weights			
Weight coefficients	0.3,0.7	0.5,0.5	0.7,0.3
Velocity (m/s)	0.18951± 0.0017	0.18953± 0.00305	0.18958± 0.00536
Angular Velocity (rad/s)	0.10003± 0.00063	0.10003± 0.00069	0.09999± 0.00242
Velocity Error Mean (m/s)	-0.01049± 0.0017	-0.01047± 0.00305	-0.01042± 0.00536
Angular Velocity Error Mean (rad/s)	3e-05± 0.00063	3e-05± 0.00069	-1e-05± 0.00242
Distance Error (m)	0.05378± 0.00265	0.0538± 0.00311	0.05382± 0.00403
Heading Error (deg.)	3.7159± 0.0831	3.71639± 0.08502	3.71371± 0.14286
RMS of position error (m)	0.07498± 0.00037	0.07496± 0.00052	0.07515± 0.00076

Table 5.9: Table of runs of the simulated distance-based method, complete version, where different weights were used for the fusing of camera and LiDAR measurements. The robots were driving in circles. The table consists of the mean± standard deviation.

camera measurements had to be chosen. Having a larger weight on the LiDAR would mean that the measurements from the LiDAR would be focused on more than the measurements from the camera. Vice versa if the larger weight was put on the camera, as shown in section 4.4. Experiments were done to determine what weights to use and to show how different weights could affect the system. Another way to pick the weights could be to use the error covariance of the measurements from the camera and LiDAR methods to weight the measurements.

The experiments to tune the weights were done with the complete versions of the simulated and physical control methods, and the robots drove in circles for 300 seconds. In the experiments, the weights that were tested were: $(w_1, w_2) = (0.3, 0.7)$, (0.5,0.5), and (0.7,0.3). All parameter values used in the experiments were chosen according to what was described earlier in this chapter for the respective control methods, with the control gains being the same as in the main experiments where the robots drove in a circle.

In table 5.9, the runs with different weights for the simulated distance-based method are shown. The weights were chosen as (0.3,0.7) for the main experiments of this method, as this was the better option because of noise and performance.

Table 5.10 shows the runs with different weights for the simulated pixel-based method. For this method, the weights were also chosen as (0.3,0.7).

Simulated pixel-based method, circle experiment. Runs with different weights			
Weight coefficients	0.3,0.7	0.5,0.5	0.7,0.3
Velocity (m/s)	0.18751± 0.00093	0.18751± 0.00248	0.18752± 0.00413
Angular Velocity (rad/s)	0.09998± 0.00595	0.09999± 0.00928	0.09989± 0.01153
Velocity Error Mean (m/s)	-0.01249± 0.00093	-0.01249± 0.00248	-0.01248± 0.00413
Angular Velocity Error Mean (rad/s)	-2e-05± 0.00595	-1e-05± 0.00928	-0.00011± 0.01153
Image coordinate n Error (pixels)	0.94477± 0.06191	0.94785± 0.15626	0.95459± 0.25336
Image coordinate m Error (pixels)	- 13.97566± 1.31241	- 13.92904± 2.00618	-13.8737± 2.49448
Distance Error (m)	0.01325± 0.00053	0.01304± 0.00109	0.01305± 0.00166
Heading Error (deg.)	1.23648± 0.10502	1.22424± 0.14999	1.22391± 0.18843
RMS of position error (m)	0.08874± 0.00047	0.08885± 0.00045	0.08891± 0.00052

Table 5.10: Table of runs of the simulated pixel-based method, complete version, where different weights were used for the fusing of camera and LiDAR measurements. The robots were driving in circles. The table consists of the mean± standard deviation.

Phys. distance-based method, circle experiment. Runs with different weights			
Weight coefficients	0.3,0.7	0.5,0.5	0.7,0.3
Velocity (m/s)	0.19495± 0.01794	0.19515± 0.01728	0.19518± 0.01931
Angular Velocity (rad/s)	0.0986± 0.00849	0.09895± 0.01188	0.09956± 0.00973
Velocity Error Mean (m/s)	-0.00505± 0.01794	-0.00485± 0.01728	-0.00482± 0.01931
Angular Velocity Error Mean (rad/s)	-0.0014± 0.00849	-0.00105± 0.01188	-0.00044± 0.00973
Distance Error (m)	0.05624± 0.00848	0.05632± 0.0091	0.05635± 0.01052
Heading Error (deg.)	6.56976± 0.45927	6.56605± 0.63645	6.60624± 0.59196
RMS of position error (m)	0.05511± 0.00255	0.05524± 0.00329	0.05539± 0.00384

Table 5.11: Table of runs of the physical distance-based method, complete version, where different weights were used for the fusing of camera and LiDAR measurements. The robots were driving in circles. The table consists of the mean± standard deviation.

As in the simulations, the weights for the complete version of the physical methods were tested to see which gave the best results. For the physical distance-based method, table 5.11 shows the results of these experiments. (0.3,0.7) were chosen as the weights for this method.

Table 5.12 shows the experiment for the physical pixel-based method. (0.7,0.3) were chosen as the weights for this method. This was because putting the weight on the LiDAR gave considerably less noise. Putting the weight on the camera here was not viable due to the amount of noise and error spikes, which in some cases caused instability. The experiment had to be done several times to obtain results for the run with the most weight on the camera. Only once did it manage to keep itself stable enough to reach 300 seconds, becoming unstable right after. There were also stability issues in the case where the LiDAR and camera sensors had equal weight, but it was more stable than putting the weight on the camera.

Phys. pixel-based method, circle experiment. Runs with different weights			
Weight coefficients	0.3,0.7	0.5,0.5	0.7,0.3
Velocity (m/s)	0.18804± 0.02095	0.18797± 0.01569	0.188± 0.0101
Angular Velocity (rad/s)	0.09809± 0.01973	0.0991± 0.0189	0.09876± 0.01654
Velocity Error Mean (m/s)	-0.01196± 0.02095	-0.01203± 0.01569	-0.012± 0.0101
Angular Velocity Error Mean (rad/s)	-0.00191± 0.01973	-0.0009± 0.0189	-0.00124± 0.01654
Image coordinate n Error (pixels)	6.59268± 1.66017	6.4404± 1.18996	6.37026± 0.82686
Image coordinate m Error (pixels)	- 30.93663± 6.33033	- 31.21646± 6.15724	- 31.21432± 5.71072
Distance Error (m)	0.07678± 0.02353	0.09354± 0.01703	0.09565± 0.0109
Heading Error (deg.)	2.59945± 0.51586	2.89439± 0.51532	3.10458± 0.4652
RMS of position error (m)	0.10177± 0.00401	0.10215± 0.00346	0.09947± 0.0028

Table 5.12: Table of runs of the physical pixel-based method, complete version, where different weights were used for the fusing of camera and LiDAR measurements. The robots were driving in circles. The table consists of the mean± standard deviation.

5.6 Simulation results and sensor characteristics

In this section, the results of the simulated main experiments will be shown. The main experiments used the parameters described in section 5.5 for the experiments and the simulated distance-based and pixel-based controllers. In addition, the noise characterization and measurement accuracy of the LiDAR and camera with marker detection was checked.

5.6.1 Sensor noise and measurement accuracy

The measurements and noise from the sensors in the simulations and physical setup were checked. Usually, in simulations, the user defines the noise models of the sensors. This was not the case in the simulations in this thesis. The LiDAR had predefined noise, and the aruco_detect package had noise, as mentioned in section 5.1. The accuracy of the sensors was measured by checking how well the sensors measured the desired distance, bearing, and pixel coordinates while the follower was stationary in the desired position behind the leader. As mentioned earlier, the desired distance was 0.75 meters, desired bearing was 0 degrees and desired pixel coordinates were $n_{des} = 199.5$, $m_{des} = 320$ for the simulations and $n_{des} = 187.2$ and $m_{des} = 315.26$ for the physical setup.

The follower was placed 0.75 meters directly behind the leader while measurements with the sensors were taken for 50 seconds. The measurements were only taken for 50 seconds because the Turtlebots in the simulations would very slowly rotate if they stood still for too long. This was because the wheel models of the robots caused integration errors to accumulate. The rotation could affect the measurement. When simulations were being done at an early stage, the size of the marker's margin was 1 cm. It was later increased to 3 cm to make the marker detection more robust. The measurement means and noise for both margin sizes will be shown in the tables for the simulation measurements, in addition to LiDAR measurements. Measurement results for the physical sensors will also be shown in tables.

For the simulations, the measurements for the distance-based method are shown in table 5.13, and the measurements for the pixel-based method are shown in table 5.14. For the physical setup, the distance-based measurements are shown in table 5.15, and the pixel-based measurements are shown in table 5.16.

Measurements from simulation setup, distance-based measurements		
Measurements	Distance (m)	Bearing (deg.)
Camera measurements, margin 3 cm	0.74963± 0.0001774	0.170078± 0.0033125
Camera measurements, margin 1 cm	0.741866± 6.59581e-05	0.197116± 0.00466967
LiDAR measurements	0.7516748± 0.0024659	-0.54408± 0.00014

Table 5.13: Measurements from the distance-based method in the simulation setup. The follower was standing still 0.75 meters directly behind the leader, and measurements were taken for 50 seconds. The table consists of the mean± standard deviation.

Measurements from simulation setup, pixel-based measurements		
Measurements	N-coordinate (pixels)	M-coordinate (pixels)
Camera measurements, margin 3 cm	199.79352± 0.0105617	319.13306± 0.0234778
Camera measurements, margin 1 cm	199.21905± 0.00669845	318.83617± 0.038889
LiDAR measurements	199.60152± 0.130541	325.84967± 0.001483

Table 5.14: Measurements from the pixel-based method in the simulation setup. The follower was standing still 0.75 meters directly behind the leader, and measurements were taken for 50 seconds. The table consists of the mean± standard deviation.

Measurements from physical setup, distance-based measurements		
Measurements	Distance (m)	Bearing (deg.)
Camera measurements	0.750319± 5.6103e-05	-0.00429± 0.000955
LiDAR measurements	0.72789± 0.0041311	-1.49757± 0.506110

Table 5.15: Measurements from the distance-based method in the physical setup. The follower was standing still 0.75 meters directly behind the leader, and measurements were taken for 50 seconds. The table consists of the mean± standard deviation.

Measurements from physical setup, pixel-based measurements		
Measurements	N-coordinate (pixels)	M-coordinate (pixels)
Camera measurements	187.19703± 0.06897	315.709029± 0.04174
LiDAR measurements	185.81204± 0.28944	331.36208± 3.00037

Table 5.16: Measurements from the pixel-based method in the physical setup. The follower was standing still 0.75 meters directly behind the leader, and measurements were taken for 50 seconds. The table consists of the mean± standard deviation.

5.6.2 Main simulation results - Distance-based method

Table 5.17 shows the simulation results of the four different versions of the distance-based method in the experiment where the robots drove in circles. Table 5.18 shows the simulation results of the different versions of the method in the experiment where they drove in a line with constant velocity. Table 5.19 shows the simulation results of the different versions of the method in the experiment where they drove in a figure-8. Table 5.20 shows the simulation results of the different versions of the method in the experiment where they drove in a line with dynamic velocity.

The camera-only and filtered versions used the camera to gather data, and the LiDAR-only version used the LiDAR to gather data. The complete version used a fusion of the sensors with the weight on the camera to gather data. A node was used to gather the position data.

Sim. distance-based method, robots driving in circle. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.18941± 0.00153	0.19012± 0.01915	0.18941± 0.0011	0.18951± 0.0017
Angular Velocity (rad/s)	0.09999± 0.00063	0.0997± 0.01656	0.10003± 0.00059	0.10003± 0.00063
Velocity Error Mean (m/s)	-0.01059± 0.00153	-0.00988± 0.01915	-0.01059± 0.0011	-0.01049± 0.0017
Angular Velocity Error Mean (rad/s)	-1e-05± 0.00063	-0.0003± 0.01656	3e-05± 0.00059	3e-05± 0.00063
Distance Error (m)	0.05373± 0.00244	0.05401± 0.00661	0.05374± 0.00249	0.05378± 0.00265
Heading Error (deg.)	3.71457± 0.08177	3.67657± 0.53755	3.71594± 0.08369	3.7159± 0.0831
RMS of position error (m)	0.0757± 0.00034	0.07355± 0.00202	0.07569± 0.00034	0.07498± 0.00037

Table 5.17: Table of runs of the simulated distance-based method, with all versions of the method and in the experiment where the robots were driving in circles. The table consists of the mean± standard deviation.

Sim. distance-based method, robots driving in a line with constant velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.20009± 0.00194	0.20026± 0.01422	0.20009± 0.00079	0.2001± 0.00312
Angular Velocity (rad/s)	1e-05± 0.00013	0.00077± 0.00441	-5e-05± 3e-05	0.0± 0.00029
Velocity Error Mean (m/s)	9e-05± 0.00194	0.00026± 0.01422	9e-05± 0.00079	0.0001± 0.00312
Angular Velocity Error Mean (rad/s)	1e-05± 0.00013	0.00077± 0.00441	-5e-05± 3e-05	0.0± 0.00029
Distance Error (m)	0.04472± 0.00249	0.04479± 0.00433	0.04471± 0.00245	0.04475± 0.00284
Heading Error (deg.)	0.00143± 0.02098	0.1819± 0.6809	-0.0085± 0.00687	0.00139± 0.07283
RMS of position error (m)	0.01803± 0.00109	0.021± 0.00261	0.018± 0.00122	0.01833± 0.00135

Table 5.18: Table of runs of the simulated distance-based method, with all versions of the method and in the experiment where the robots were driving in a line with constant velocity. The table consists of the mean± standard deviation.

Sim. distance-based method, robots driving in figure-8. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-0.01045± 0.00369	-0.01116± 0.01653	-0.01043± 0.00343	-0.01064± 0.00413
Angular Velocity Error Mean (rad/s)	0.01007± 0.03488	0.01119± 0.05369	0.01114± 0.03774	0.01122± 0.03781
Distance Error (m)	0.05502± 0.00398	0.05487± 0.00673	0.05501± 0.00396	0.055± 0.00434
Heading Error (deg.)	0.0105± 2.78071	0.12842± 2.8346	0.02081± 2.7888	0.02728± 2.78873
RMS of position error (m)	0.07882± 0.01417	0.08602± 0.01911	0.07892± 0.01416	0.08064± 0.01501

Table 5.19: Table of runs of the simulated distance-based method, with all versions of the method and in the experiment where the robots were driving in a figure-8. The table consists of the mean± standard deviation.

Sim. distance-based method, robots driving in a line with dynamic velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	6e-05± 0.00811	8e-05± 0.01202	0.00014± 0.00915	3e-05± 0.01145
Angular Velocity Error Mean (rad/s)	0.00015± 0.00053	0.00153± 0.00375	4e-05± 0.0005	1e-05± 0.00049
Distance Error (m)	0.04885± 0.01497	0.04892± 0.01531	0.04896± 0.01552	0.04893± 0.01533
Heading Error (deg.)	0.01657± 0.05841	0.18476± 0.43062	0.00415± 0.05745	0.00186± 0.07422

Table 5.20: Table of runs of the simulated distance-based method, with all versions of the method and in the experiment where the robots were driving in a line with dynamic velocity. The table consists of the mean± standard deviation.

Sim. pixel-based method, robots driving in circle. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.18739± 0.00055	0.18762± 0.0105	0.18739± 0.00035	0.18751± 0.00093
Angular Velocity (rad/s)	0.1± 0.00116	0.09906± 0.03093	0.09999± 0.00118	0.09998± 0.00595
Velocity Error Mean (m/s)	-0.01261± 0.00055	-0.01238± 0.0105	-0.01261± 0.00035	-0.01249± 0.00093
Angular Velocity Error Mean (rad/s)	-0.0± 0.00116	-0.00094± 0.03093	-1e-05± 0.00118	-2e-05± 0.00595
Image coordinate n Error (pixels)	0.94025± 0.02093	0.96668± 0.34819	0.94013± 0.02119	0.94477± 0.06191
Image coordinate m Error (pixels)	- 14.00601± 0.51863	- 13.63514± 3.63932	- 14.00611± 0.55083	- 13.97566± 1.31241
Distance Error (m)	0.01405± 0.00025	0.01844± 0.00669	0.01405± 0.00025	0.01325± 0.00053
Heading Error (deg.)	1.19238± 0.05058	1.26799± 0.33833	1.19253± 0.0535	1.23648± 0.10502
RMS of position error (m)	0.08955± 0.00047	0.08876± 0.00045	0.08956± 0.00049	0.08874± 0.00047

Table 5.21: Table of runs of the simulated pixel-based method, with all versions of the method and in the experiment where the robots were driving in circles. The table consists of the mean± standard deviation.

5.6.3 Main simulation results - Pixel-based method

Table 5.21 shows the simulation results of the four different versions of the pixel-based method in the experiment where the robots drove in circles. Table 5.22 shows the simulation results of the different versions of the method in the experiment where the robots drove in a line with constant velocity. Table 5.23 shows the simulation results of the different versions of the method in the experiment where the robots drove in a figure-8. Table 5.24 shows the simulation results of the different versions of the method in the experiment where the robots drove in a line with dynamic velocity.

The camera-only and filtered versions used the camera to gather data and the LiDAR-only version used the LiDAR to gather data. The complete version used a fusion of the sensors with the weight on the camera to gather data. A node was used to gather the position data.

Sim. pixel-based method robots driving in a line with constant velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.2± 0.0005	0.20004± 0.00362	0.2± 0.00028	0.2± 0.00061
Angular Velocity (rad/s)	-5e-05± 6e-05	0.00019± 0.0012	-5e-05± 2e-05	-5e-05± 0.00049
Velocity Error Mean (m/s)	0.0± 0.0005	4e-05± 0.00362	0.0± 0.00028	0.0± 0.00061
Angular Velocity Error Mean (rad/s)	-5e-05± 6e-05	0.00019± 0.0012	-5e-05± 2e-05	-5e-05± 0.00049
Image coordinate n Error (pixels)	1.40473± 0.02388	1.41077± 0.14581	1.40476± 0.02422	1.40504± 0.04852
Image coordinate m Error (pixels)	0.03961± 0.04561	-0.14143± 0.89725	0.04139± 0.03739	0.03849± 0.70822
Distance Error (m)	0.02048± 0.00037	0.02692± 0.0029	0.0205± 0.00035	0.02124± 0.00053
Heading Error (deg.)	0.08737± 0.00508	0.01315± 0.08345	0.08733± 0.00437	0.2871± 0.01345
RMS of position error (m)	0.00624± 0.00047	0.00921± 0.00117	0.00624± 0.00029	0.00612± 0.00101

Table 5.22: Table of runs of the simulated pixel-based method, with all versions of the method and in the experiment where the robots were driving in a line with constant velocity. The table consists of the mean± standard deviation.

Sim. pixel-based method, robots driving in figure-8. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-0.01127± 0.00319	-0.01188± 0.00967	-0.01126± 0.0032	-0.01145± 0.00359
Angular Velocity Error Mean (rad/s)	0.01209± 0.03942	0.01129± 0.04694	0.01161± 0.03868	0.01091± 0.03739
Image coordinate n Error (pixels)	0.99152± 0.11489	0.98524± 0.32615	0.99239± 0.12126	0.987± 0.15209
Image coordinate m Error (pixels)	3.16369± 13.43758	3.21596± 13.66066	3.33295± 13.44139	3.43382± 13.42344
Distance Error (m)	0.01459± 0.00182	0.01879± 0.00628	0.01462± 0.00191	0.01443± 0.00258
Heading Error (deg.)	-0.14909± 1.07047	-0.29909± 1.27032	-0.16442± 1.07174	0.0394± 0.92813
RMS of position error (m)	0.08302± 0.01445	0.08885± 0.01607	0.08304± 0.0145	0.08486± 0.01478

Table 5.23: Table of runs of the simulated pixel-based method, with all versions of the method and in the experiment where the robots were driving in a figure-8. The table consists of the mean± standard deviation.

Sim. pixel-based method, robots driving in a line with dynamic velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	8e-05± 0.01087	-4e-05± 0.01128	-3e-05± 0.01177	-6e-05± 0.01098
Angular Velocity Error Mean (rad/s)	-5e-05± 0.00057	0.00038± 0.00732	-6e-05± 0.00042	-4e-05± 0.00074
Image coordinate n Error (pixels)	0.11632± 1.3737	0.11527± 1.37931	0.12369± 1.40248	0.12025± 1.39401
Image coordinate m Error (pixels)	0.01848± 0.21642	-0.14039± 2.77977	0.02427± 0.20046	0.01497± 0.4572
Distance Error (m)	-0.00234± 0.02506	0.00289± 0.02626	-0.00209± 0.02542	-0.00076± 0.02553
Heading Error (deg.)	0.0907± 0.02132	0.01306± 0.25855	0.08996± 0.01995	0.31321± 0.05044

Table 5.24: Table of runs of the simulated pixel-based method, with all versions of the method and in the experiment where the robots were driving in a line with dynamic velocity. The table consists of the mean± standard deviation.

5.7 Physical setup results

In this section the results of the physical main experiments will be shown. The main experiments used the parameters described in section 5.5 for the experiments and the physical distance-based and pixel-based controllers.

5.7.1 Main physical setup results - Distance-based method

Table 5.25 shows the physical setup results of the four different versions of the distance-based method in the experiment where the robots drove in circles. Table 5.26 shows the physical setup results of the different versions of the method in the experiment where the robots drove in a line with constant velocity. Table 5.27 shows the physical setup results of the different versions of the method in the experiment where the robots drove in a figure-8. Table 5.28 shows the physical setup results of the different versions of the method in the experiment where the robots drove in a line with dynamic velocity.

The camera-only and filtered versions used the camera to gather data and the LiDAR-only version used the LiDAR to gather data. The complete version used a fusion of the sensors with the weight on the camera to gather data. The motion capture system was used to gather the position data.

Phys. distance-based method, robots driving in circle. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.19508± 0.02084	0.19819± 0.04028	0.19513± 0.02711	0.19495± 0.01794
Angular Velocity (rad/s)	0.09981± 0.00958	0.1114± 0.03895	0.10138± 0.016	0.0986± 0.00849
Velocity Error Mean (m/s)	-0.00492± 0.02084	-0.00181± 0.04028	-0.00487± 0.02711	-0.00505± 0.01794
Angular Velocity Error Mean (rad/s)	-0.00019± 0.00958	0.0114± 0.03895	0.00138± 0.016	-0.0014± 0.00849
Distance Error (m)	0.05625± 0.00721	0.05804± 0.01449	0.0564± 0.01196	0.05624± 0.00848
Heading Error (deg.)	6.63174± 0.45289	6.96413± 1.24441	6.65484± 0.77865	6.56976± 0.45927
RMS of position error (m)	0.05455± 0.00232	0.05719± 0.00713	0.05506± 0.00295	0.05511± 0.00255

Table 5.25: Table of runs of the physical distance-based method, with all versions of the method and in the experiment where the robots were driving in circles. The table consists of the mean± standard deviation.

Phys. distance-based method, robots driving in a line with constant velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.20103± 0.00644	0.20248± 0.02825	0.201± 0.00241	0.19932± 0.00973
Angular Velocity (rad/s)	0.00187± 0.00312	-0.00321± 0.00269	-0.00546± 0.00115	-0.00189± 0.00323
Velocity Error Mean (m/s)	0.00103± 0.00644	0.00248± 0.02825	0.001± 0.00241	-0.00068± 0.00973
Angular Velocity Error Mean (rad/s)	0.00187± 0.00312	-0.00321± 0.00269	-0.00546± 0.00115	-0.00189± 0.00323
Distance Error (m)	0.07025± 0.00406	0.07143± 0.01222	0.07017± 0.00311	0.06974± 0.00628
Heading Error (deg.)	0.86766± 1.3316	-1.03541± 0.79261	-1.75441± 0.2636	-0.65913± 1.04441
RMS of position error (m)	0.01342± 0.00344	0.02609± 0.00684	0.02724± 0.00121	0.02248± 0.00139

Table 5.26: Table of runs of the physical distance-based method, with all versions of the method and in the experiment where the robots were driving in a line with constant velocity. The table consists of the mean± standard deviation.

Phys. distance-based method, robots driving in figure-8. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-0.00694± 0.01296	-0.00732± 0.04166	-0.00737± 0.01371	-0.00755± 0.01226
Angular Velocity Error Mean (rad/s)	-0.01263± 0.03877	-0.01715± 0.05273	-0.01772± 0.05248	-0.01314± 0.04196
Distance Error (m)	0.06385± 0.00577	0.06437± 0.01686	0.06369± 0.00758	0.06369± 0.00737
Heading Error (deg.)	2.65718± 4.77272	2.49685± 4.8702	2.525± 4.71078	2.68571± 4.77692
RMS of position error (m)	0.0562± 0.00933	0.07303± 0.02096	0.05653± 0.01	0.06105± 0.01208

Table 5.27: Table of runs of the physical distance-based method, with all versions of the method and in the experiment where the robots were driving in a figure-8. The table consists of the mean± standard deviation.

Phys. distance-based method, robots driving in a line with dynamic velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-0.00057± 0.01194	0.0006± 0.02961	6e-05± 0.01568	0.0004± 0.02076
Angular Velocity Error Mean (rad/s)	-0.0022± 0.00214	-0.00352± 0.00301	-0.00329± 0.00194	-0.00246± 0.00175
Distance Error (m)	0.05469± 0.01855	0.05565± 0.02139	0.05482± 0.01936	0.05504± 0.02024
Heading Error (deg.)	-0.70035± 0.67495	-1.07073± 0.84001	-1.04732± 0.5843	-0.79956± 0.54723

Table 5.28: Table of runs of the physical distance-based method, with all versions of the method and in the experiment where the robots were driving in a line with dynamic velocity. The table consists of the mean± standard deviation.

Phys. pixel-based method, robots driving in circle. Runs with different versions of method			
Methods	Camera-only	Lidar-only	Complete
Velocity (m/s)	0.18633± 0.032	0.18621± 0.01734	0.188± 0.0101
Angular Velocity (rad/s)	0.09801± 0.01545	0.0997± 0.02786	0.09876± 0.01654
Velocity Error Mean (m/s)	-0.01367± 0.032	-0.01379± 0.01734	-0.012± 0.0101
Angular Velocity Error Mean (rad/s)	-0.00199± 0.01545	-0.0003± 0.02786	-0.00124± 0.01654
Image coordinate n Error (pixels)	6.67304± 2.11631	6.30239± 1.05067	6.37026± 0.82686
Image coordinate m Error (pixels)	- 31.24326± 3.95961	- 31.25168± 6.99737	- 31.21432± 5.71072
Distance Error (m)	0.11118± 0.03132	0.11375± 0.02179	0.09565± 0.0109
Heading Error (deg.)	2.29336± 0.30957	2.7565± 0.61643	3.10458± 0.4652
RMS of position error (m)	0.11442± 0.00734	0.1205± 0.00318	0.09947± 0.0028

Table 5.29: Table of runs of the physical pixel-based method, with all versions of the method except the filtered version since it could not be kept stable, in the experiment where the robots were driving in circles. The table consists of the mean± standard deviation.

5.7.2 Main physical setup results - Pixel-based method

Table 5.29 shows the physical setup results of three different versions of the pixel-based method in the experiment where the robots drove in circles. The filtered version could not be kept stable for 300 seconds, so it is not in the results. Table 5.30 shows the physical setup results of the pixel-based method in the experiment where the robots drove in a line with constant velocity. Table 5.31 shows the physical setup results of the method in the experiment where the robots drove in a figure-8. Table 5.32 shows the physical setup results of the method in the experiment where the robots drove in a line with dynamic velocity.

The camera-only and filtered versions used the camera to gather data and the LiDAR-only version used the LiDAR to gather data. The complete version used a fusion of the sensors with the weight on the LiDAR to gather data but used the camera to gather the distance and heading error data. The motion capture system was used to gather the position data.

Phys. pixel-based method, robots driving in a line with constant velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity (m/s)	0.20423± 0.02804	0.19981± 0.00586	0.20089± 0.03316	0.2011± 0.01307
Angular Velocity (rad/s)	-0.00544± 0.00224	-0.00144± 0.00291	-0.00052± 0.0037	-0.00543± 0.00212
Velocity Error Mean (m/s)	0.00423± 0.02804	-0.00019± 0.00586	0.00089± 0.03316	0.0011± 0.01307
Angular Velocity Error Mean (rad/s)	-0.00544± 0.00224	-0.00144± 0.00291	-0.00052± 0.0037	-0.00543± 0.00212
Image coordinate n Error (pixels)	7.32053± 2.07063	6.52588± 0.32598	8.08468± 3.07582	6.65101± 0.88808
Image coordinate m Error (pixels)	9.64806± 3.4785	3.27375± 4.61559	1.68464± 6.21136	9.60065± 4.70317
Distance Error (m)	0.08032± 0.03307	0.11684± 0.00665	0.13586± 0.03918	0.07608± 0.01482
Heading Error (deg.)	-0.75768± 0.28064	-0.28899± 0.40744	-0.13181± 0.4944	-0.10727± 0.42827
RMS of position error (m)	0.04716± 0.01921	0.02348± 0.00075	0.02312± 0.00822	0.03764± 0.00393

Table 5.30: Table of runs of the physical pixel-based method, with all versions of the method and in the experiment where the robots were driving in a line with constant velocity. The table consists of the mean± standard deviation.

Phys. pixel-based method, robots driving in figure-8. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-0.01251± 0.03331	-0.01376± 0.0155	-0.01305± 0.02964	-0.01145± 0.0115
Angular Velocity Error Mean (rad/s)	-0.01168± 0.03906	-0.01473± 0.05176	-0.01323± 0.05197	-0.01403± 0.04442
Image coordinate n Error (pixels)	6.80088± 2.21669	6.28766± 0.92685	6.93126± 2.46197	6.42332± 0.8825
Image coordinate m Error (pixels)	-6.73295± 30.96185	-5.80126± 30.94373	-6.29257± 30.90391	-6.16991± 30.94994
Distance Error (m)	0.11005± 0.03936	0.11329± 0.01907	0.11257± 0.0418	0.08301± 0.01618
Heading Error (deg.)	0.49288± 2.28187	0.51197± 2.72897	0.46134± 2.28749	1.27611± 2.34317
RMS of position error (m)	0.09825± 0.02548	0.10943± 0.0171	0.10055± 0.02666	0.08647± 0.01531

Table 5.31: Table of runs of the physical pixel-based method, with all versions of the method and in the experiment where the robots were driving in a figure-8. The table consists of the mean± standard deviation.

Phys. pixel-based method, robots driving in line with dynamic velocity. All versions				
Methods	Camera-only	Lidar-only	Filtered	Complete
Velocity Error Mean (m/s)	-5e-05± 0.03487	0.00063± 0.01569	0.00095± 0.03504	0.00111± 0.01787
Angular Velocity Error Mean (rad/s)	-0.002± 0.00145	-0.00211± 0.00276	-0.00202± 0.00154	-0.00146± 0.00183
Image coordinate n Error (pixels)	5.24912± 2.78029	5.0401± 2.033	5.456± 3.1478	5.0728± 2.13496
Image coordinate m Error (pixels)	4.01349± 2.19263	4.14988± 4.13699	4.03559± 2.59323	3.18234± 3.84937
Distance Error (m)	0.04791± 0.05328	0.08894± 0.03942	0.04657± 0.05899	0.04566± 0.04188
Heading Error (deg.)	-0.3111± 0.17433	-0.36633± 0.36519	-0.31496± 0.20543	0.62514± 0.21384

Table 5.32: Table of runs of the physical pixel-based method, with all versions of the method and in the experiment where the robots were driving in a line with dynamic velocity. The table consists of the mean± standard deviation.

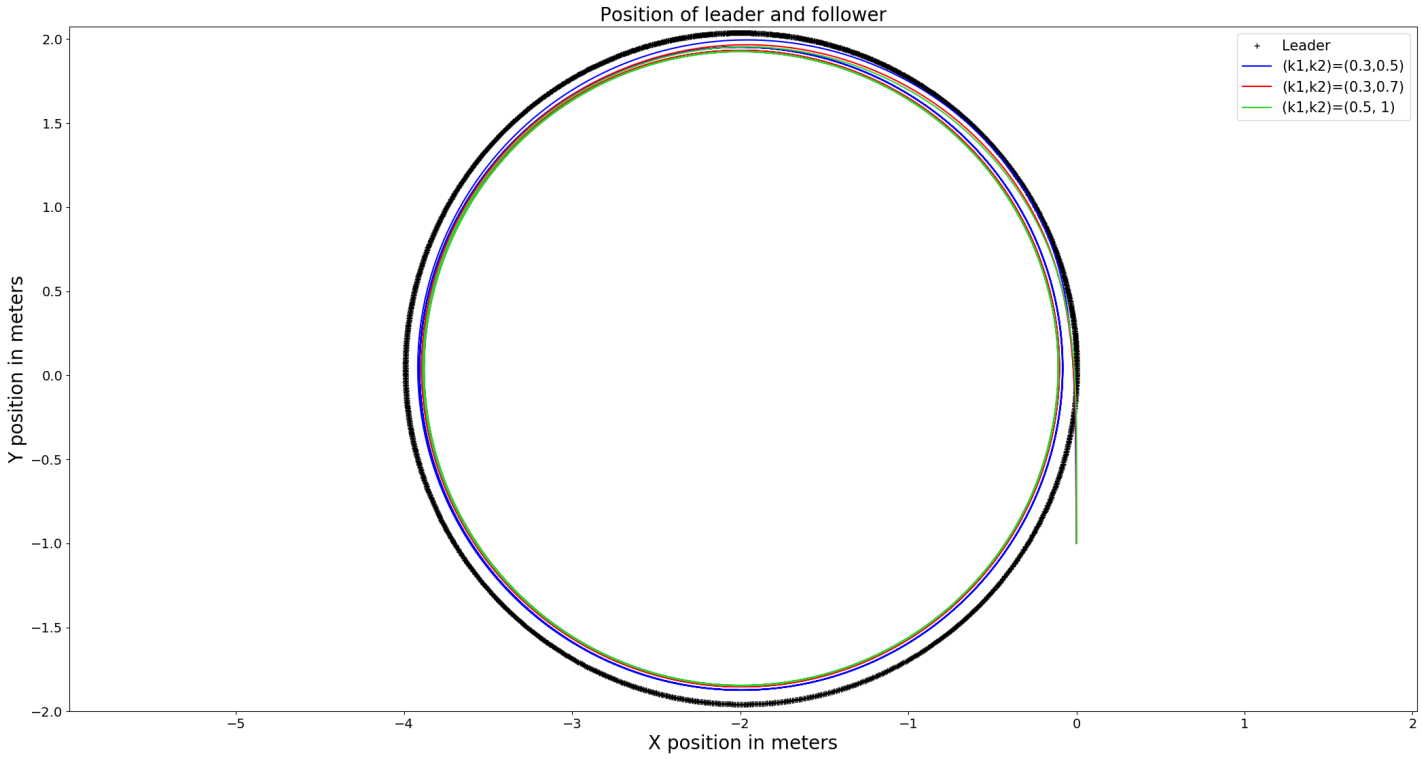

Figure 5.3: A plot of the position of the leader and follower in the simulated distance-based method experiment where the robots drove in circles with different control gains. The y-position in meters is on the y-axis, and the x-position in meters is on the x-axis.

5.8 Plots

In addition to the tables, some plots will be presented to better show different aspects of the controllers. Figures 5.3-5.5 show the positions of the Turtlebots in different experiments with the simulated distance-based method. Figures 5.6, 5.7, and 5.9 show plots of the RMS of the position error in different experiments. Figure 5.6 is from the simulation, and 5.7 and 5.9 are from the physical experiments. Plot 5.8 shows the velocity and angular velocity in the experiment where the robots drove in a line with constant velocity with the pixel-based method in the physical setup.

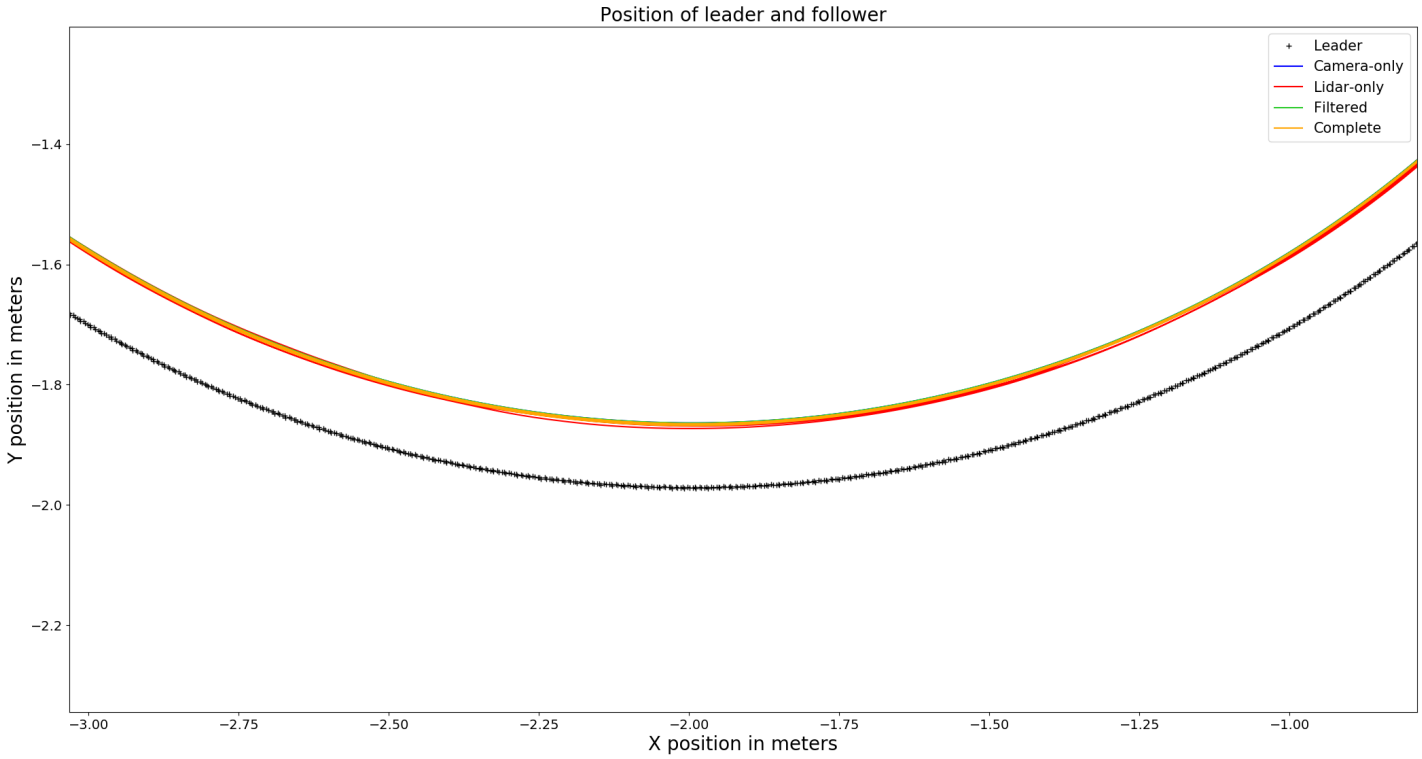

Figure 5.4: A zoomed-in plot of the position of the follower and leader in the simulated distance-based method experiment where the robots drove in circles, with all versions of the method. y position in meters is given on the y-axis, and x position in meters is given on the x-axis.

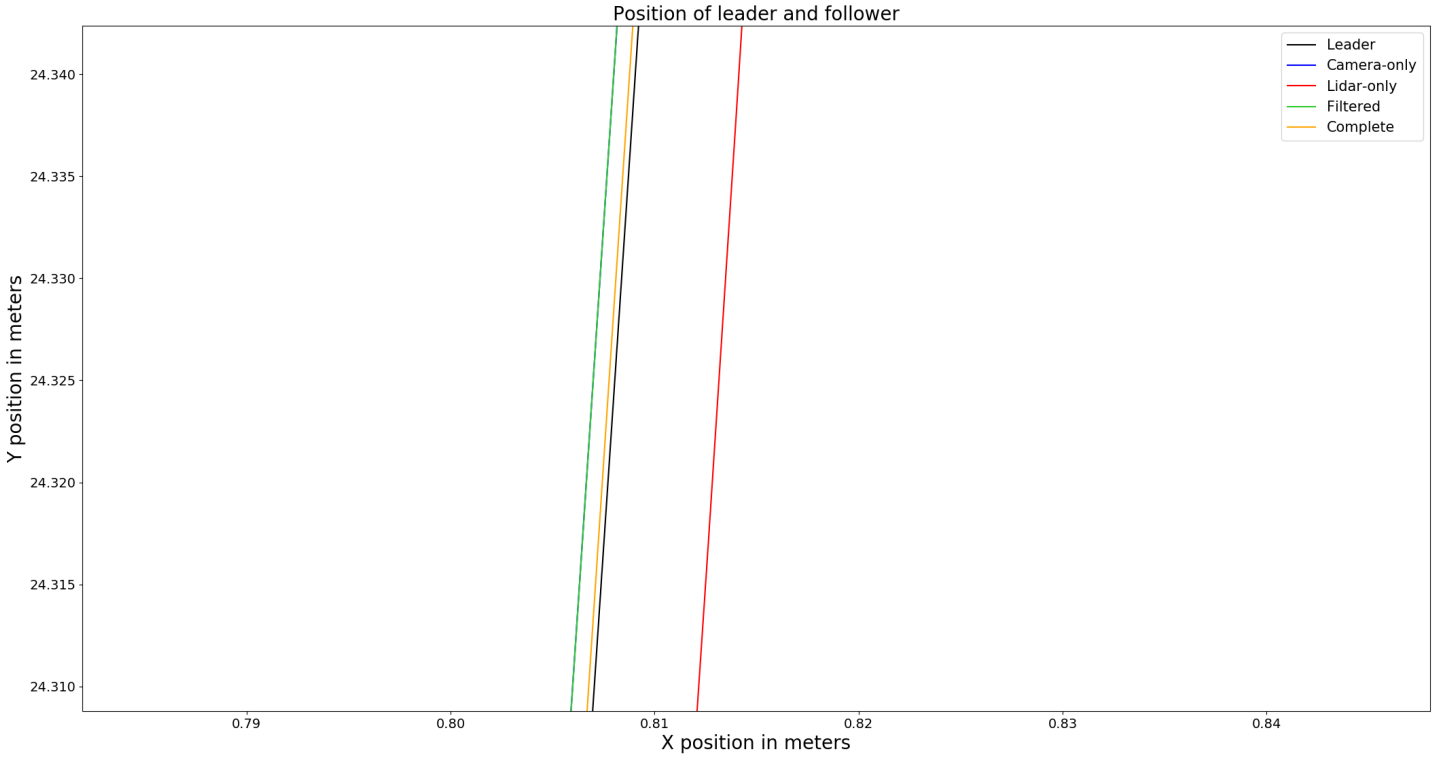

Figure 5.5: A zoomed-in plot of the position of the follower and leader in the simulated distance-based method experiment where the robots drove in a line with constant velocity, with all versions of the method. y position in meters is given on the y-axis, and x position in meters is given on the x-axis.

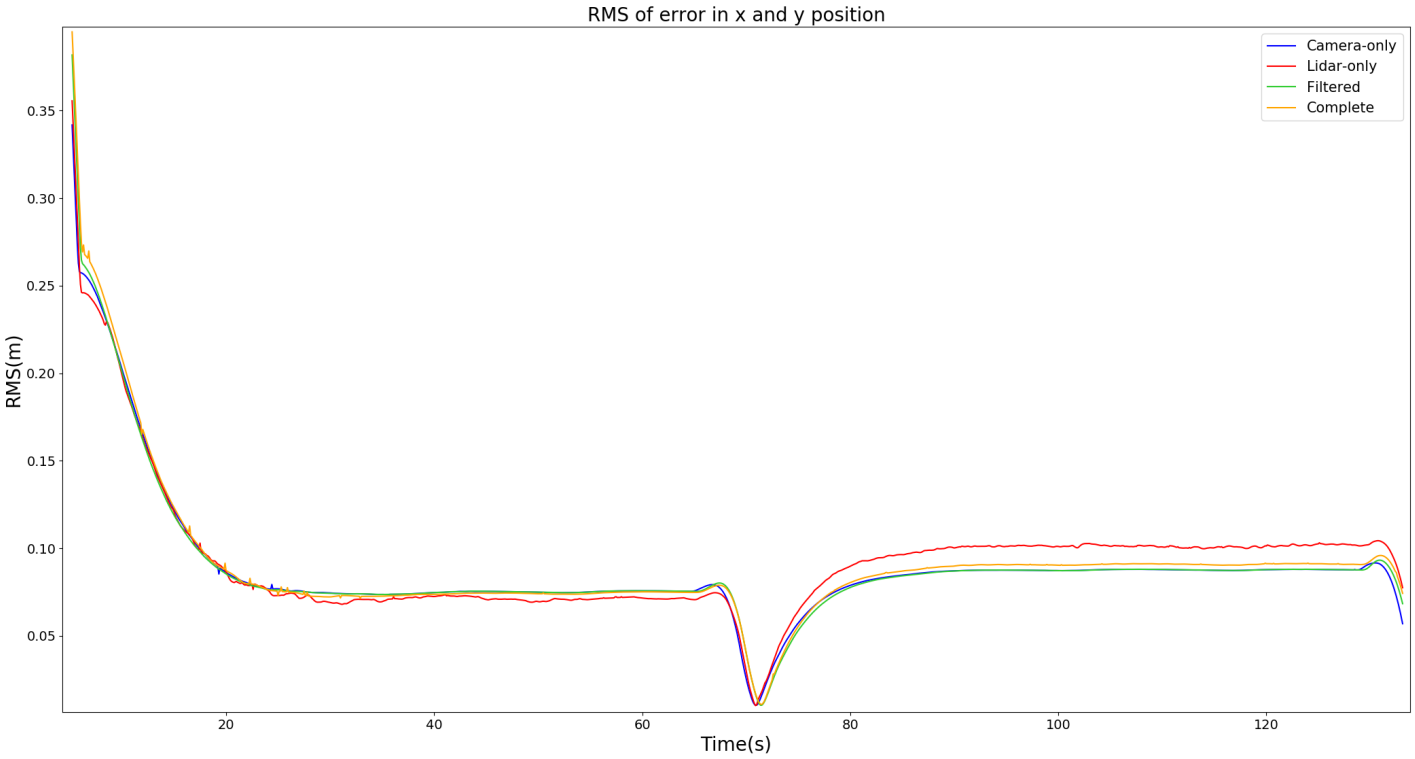

Figure 5.6: A plot of the RMS of the position error in the simulated distance-based method experiment where the robots drove in a figure-8, with all the versions of the method. The RMS of the position error in meters is shown on the y-axis, and the time in seconds is shown on the x-axis.

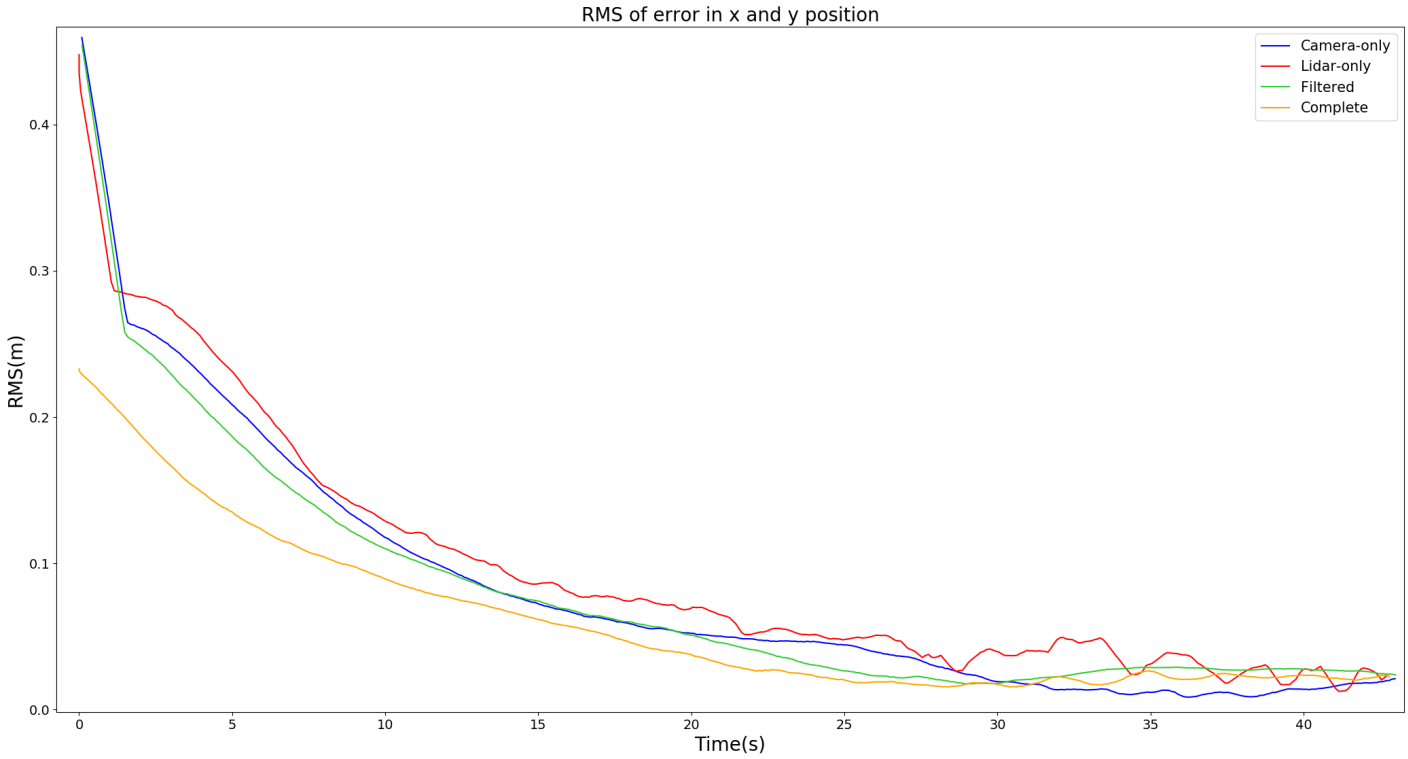

Figure 5.7: A plot of the RMS of the position error in the physical distance-based method experiment where the robots drove in a line with constant velocity, with all the versions of the method. The RMS of the position error in meters is shown on the y-axis, and the time in seconds is shown on the x-axis.

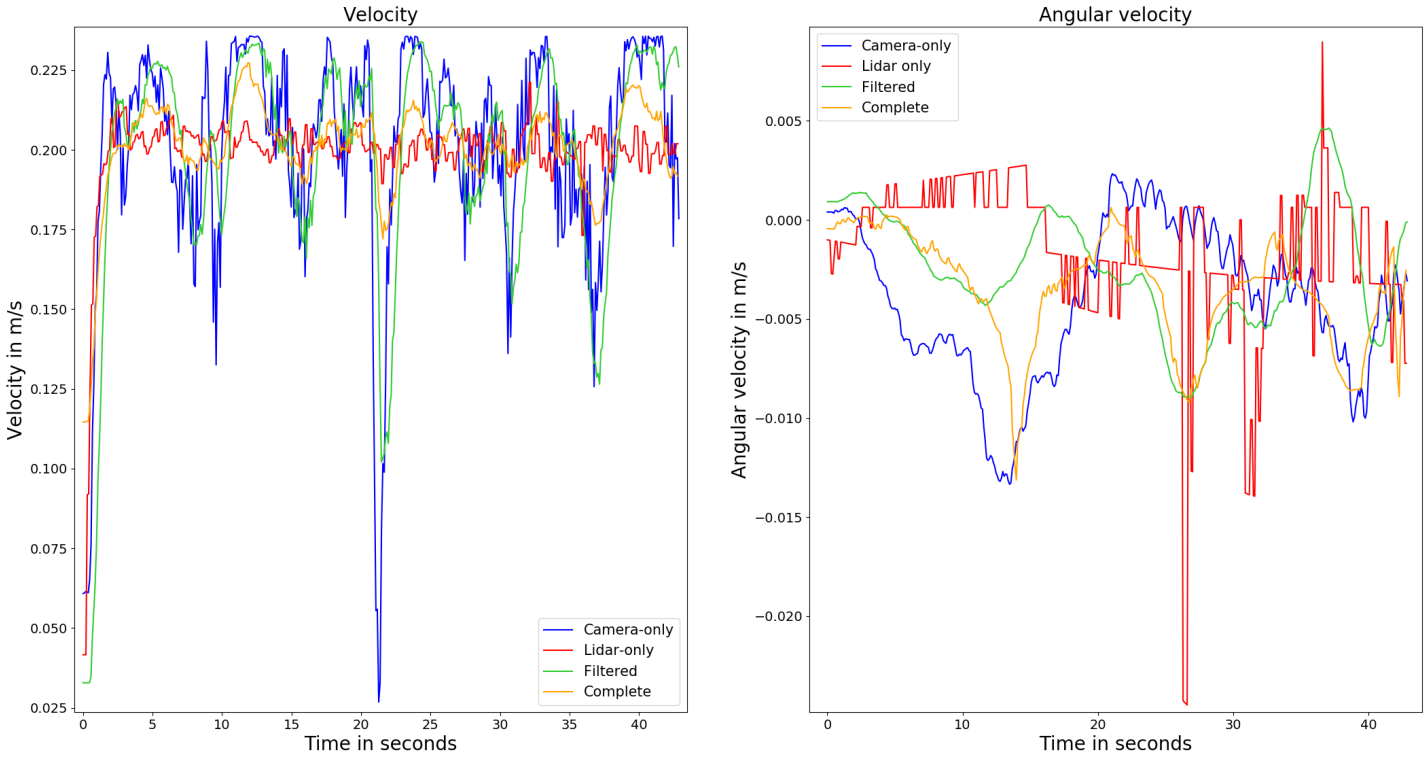

Figure 5.8: A plot of the velocity and angular velocity of the different versions in the physical pixel-based method experiment where the robots drove in a line with constant velocity. The plot to the left shows the velocity. It has velocity in m/s on the y-axis and time in seconds on the x-axis. The plot to the right shows the angular velocity. It has rad/s on the y-axis and seconds on the x-axis.

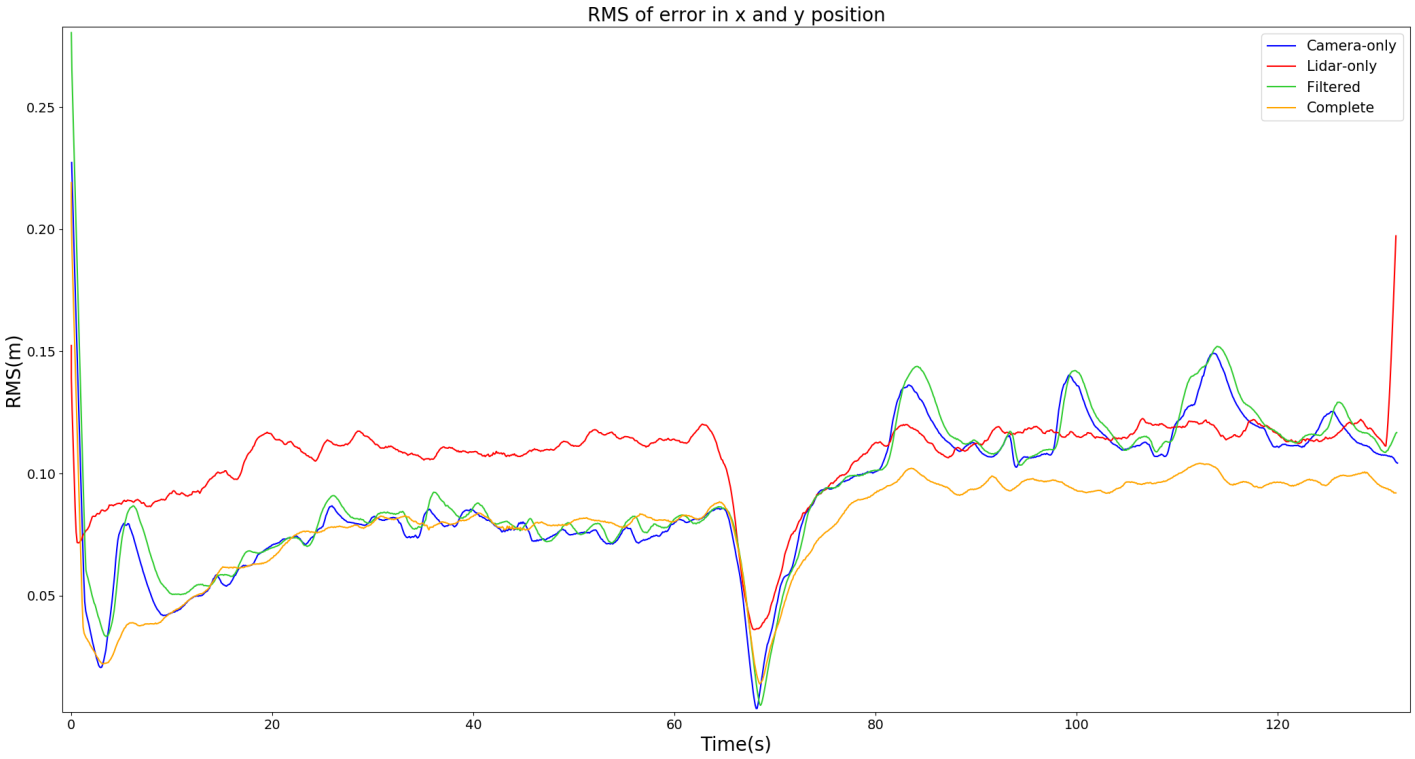

Figure 5.9: A plot of the RMS of the position error in the physical pixel-based method experiment where the robots drove in a figure-8, with all the versions of the method. The RMS of the position error in meters is shown on the y-axis, and the time in seconds is shown on the x-axis.

Chapter 6

Discussion

In this chapter, the results of the experiments will be discussed. The controllers and the different versions of the controllers will be compared. The results of the controllers will also be compared to the results in (Verginis et al., 2015) and (Miao et al., 2021).

6.1 Simulations

In this section, the results of the simulated experiments will be discussed, as well as the results of the sensor accuracy check in the simulations and simulated experiments regarding the tuning of parameters. When discussing the experiments, the performance metrics from section 5.4.1 are used. When comparing the methods and controllers, the RMS of the position error will be considered separately from the other errors. When speaking of noise, it means the standard deviation of these errors.

6.1.1 Characteristics of the simulated sensors

In table 5.13, the mean and standard deviation of the distance and bearing measurements of the camera and LiDAR in the simulations were checked. This was done with the camera on a marker with a 3 cm margin, a marker with a 1 cm margin, and the LiDAR. This was done to check the accuracy and noise of the sensors in the simulations. Accuracy was measured by checking how well the sensor measured the desired distance, bearing, and pixel coordinates. As previously mentioned, the packages in ROS set a default noise model for the simulated LiDAR to model the physical LiDAR. In addition, the marker detection also had noise.

In the test in table 5.13, the camera measurements with the 1 cm margin had the least

noise when it came to the distance measurements, but the mean was not close to what it should be. It was almost 10 centimeters off from the desired mean of 0.75 meters. This was because the detection package had issues detecting corners when using a small margin, making the detection unreliable. The LiDAR and camera measurements with a 3 cm margin were closer to the desired mean. When measuring the bearing, the camera did well with both margin sizes. The LiDAR measured a mean bearing angle of around -0.5 degrees, while the target was 0 degrees. This was most likely because of the LiDAR resolution since it had an angular resolution of 1 degree. A LiDAR with better angular resolution would likely have given more accurate angle measurements. In addition, while the camera only measured the marker, the LiDAR measured everything within a cone of detection. This means that the LiDAR measured the entire back of the leader.

The results of the sensor test for the pixel measurements are shown in table 5.14. The m-coordinate measurements from the camera with both margin sized were accurate, with an error of around one pixel. The LiDAR missed the target of 320 pixels by almost 6 pixels. This was because the bearing measurements of the LiDAR were used to estimate the m-coordinates with the LiDAR. If the bearing measurements are inaccurate, the m-coordinate measurements will be inaccurate too. In both table 5.13 and table 5.14, the LiDAR had the least noise when it came to measuring the bearing and m-coordinates but the most when measuring distance and n-pixel coordinates. This could be a result of the LiDAR having a high resolution and being very accurate in the distance measurements but having a low resolution in the bearing measurements, giving it less noise in the bearing measurements while the robots were stationary.

Since the measurements were taken while the robots were stationary, they give an idea of the measurement noise and accuracy of the sensors. But the sensors might perform differently when the robots move. For example, using the 1 cm margin caused the marker detection to be less reliable and fail during the experiments. Therefore, it was important to use a big enough margin so that the ArUco detection could detect the corners of the marker properly.

6.1.2 Effects of the maximum steady-state errors, weights, and control gains on the simulated system

Table 5.7 shows the results of the experiment where the simulated distance-based LiDAR version was using different maximum steady-state errors $\rho_{d,\infty}$ in the circle experiment. The errors in the velocity and distance became smaller if $\rho_{d,\infty}$ was smaller. This is because the $\rho_{d,\infty}$ affects the linear control and is the maximum steady-state error for the distance error. Pushing $\rho_{d,\infty}$ down increased the linear control performance. The performance in the angular controller, which affected the angular velocity and heading errors, was about the same in both runs. This was because $\rho_{\beta,\infty}$ was kept constant. When $\rho_{d,\infty}$ was decreased, the system noise increased. This means that performance is gained at the cost of stability. Noise can be

reduced by adjusting the control gain, but this might also change the performance. When tuning the boundaries, the gain must be tuned along with them. Lowering $\rho_{d,\infty}$ lowers the position error, as the distance error is decreased.

Table 5.8 shows the results of using different control gains in the distance-based method for the circle experiment. Having larger gains gave more error in the velocities but less error in the distance and heading. This can be seen when comparing the run with $(k_d, k_\beta) = (0.5, 1)$ as gains to the others. Higher gains gave more noise due to more oscillations in the system. The results of the runs with (0.3,0.5) and (0.3,0.7) as gains had similar distance error and distance noise. This was similar because they both used $k_d = 0.3$. The run with (0.3,0.7) had a lower heading error due to having a higher k_β. Lower gains gave less position error. This was likely connected to the heading error, as the run with (0.5,1) had less distance error, but the position error was higher.

The position of the robots in the runs with different control gains is shown in figure 5.3. The lower gain runs followed the circle made by the leader more closely, while the runs with higher gains were closer to the middle of the circle. It was as if they were taking more of a shortcut, and this led to a higher position error. The shortcutting was likely connected to the heading error.

Tables 5.9 and 5.10 show the results of the experiments where the robots drove in a circle with different weights on the complete version of the distance-based and pixel-based methods. The runs with the different weights in the distance-based method had very similar errors overall, and the differences in errors were almost negligible. The same also applies to the errors between the runs in the pixel-based method. Looking at the noise shows that putting weight on the camera gave considerably less noise than the alternatives. This applies for both control methods. The low noise was because the camera and marker detection performed very well in the simulations, with the camera having little noise. Because of this, the weight was put on the camera for both methods in the simulations.

6.1.3 Distance-based

In this section, the results of the distance-based method in the simulations will be discussed. The main experiments were the experiment where robots drove in a circle, in a line with constant velocity, in a figure-8, and a line with dynamic velocity. The experiments were done for all the versions of the method. Table 5.17 shows the results for the circle experiment, 5.18 shows the results for the experiment where the robots drove in a line with constant velocity, 5.19 shows the results for the experiment where they drove in a figure-8, and 5.20 shows the results for the experiment where they drove in a line with dynamic velocity. The respective experiments will be called the circle, constant, figure-8, and dynamic experiments from now on.

In the circle experiment, the filtered and camera versions were the best as they had the least noise and errors. They were almost identical in terms of error, but the camera version was a little better. The LiDAR version was the worst, as it had the most noise and errors. The LiDAR version had the least heading error and the least RMS of position error. The complete version also had a relatively low position error. This could be connected to these versions using a LiDAR.

In section 6.1.2, the connection between the heading error and shortcutting was mentioned. In the experiment in table 5.17, even with the heading error of the LiDAR version being smaller, the position error was smaller. This could mean that using the LiDAR sensor led to less shortcutting. In the circle experiment, shortcutting and distance errors were the main contributors to the position error. Less shortcutting would explain why the LiDAR and complete versions had more distance errors yet lower position errors in this experiment. In figure 5.4, it can be seen that the LiDAR and complete versions shortcut a little less than the camera-based versions. Shortcutting is a consequence of the follower following the leader directly instead of following the leader's trajectory. The follower cuts corners to minimize the distance and heading errors, leading to it not being on the leader's path.

In the constant experiment, the filtered version was the best as it had the least noise and error. The camera and complete versions were close to the filtered version in terms of error. The LiDAR version was the worst, as it had the most error and noise. The LiDAR version had a higher position error than the other versions. This was because the LiDAR was worse at detecting angles, making the LiDAR version struggle with driving directly behind the leader when the leader was driving in a line. Figure 5.5 shows this.

In the figure-8 experiment, the complete and camera versions were the best since they had the least error and noise respectively. In the circle experiment, the filtered version had a little less noise in the angular velocity, but a little more in the heading error. When comparing the noise of the camera and filtered versions in the figure-8 experiment, the filtered version had a little more noise in the heading and angular velocity than the camera version and a little less in the linear velocity and distance. This could be because the angular velocity changed in this experiment, leading to the filtered version having more noise in these errors. This was because of the delay introduced by the moving average filter, which made the versions that used the filter reacts slower to changes in velocity since previous velocities were used to calculate the new velocity.

The LiDAR version had the largest position error. This was because the LiDAR version had issues adjusting after the angular velocity changed, as seen in figure 5.6. The position error increased after the angular velocity switched. This was related to the LiDAR sensor since the complete version shows similar behavior to a lesser degree. The higher position error could be because the LiDAR was bad at measuring angles. It could also be because the LiDAR sensor used the entire back of the leader to measure the bearing angle and distance, not only the marker. When the angular velocity changed, the LiDAR measured the back

and might also have detected a little of the sides of the leader. This could lead to the LiDAR version shortcutting more after the switch in angular velocity.

In the dynamic experiment, the complete and camera versions were the best, as they had the least error and noise respectively. In the constant experiment, the filtered version had less noise in the distance and linear velocity. The moving average filter made it so that the filtered and complete versions of the dynamic experiment had a little more noise in the linear controller than if the velocity was constant. This was because when the leader's velocity changed, the follower tried to adjust to the new velocity, which caused oscillations. The oscillations remained in the system longer with the moving average filter. Since the velocity changed more frequently in the dynamic experiment than in the figure-8 experiment, the effect of the moving average filter's delay on the system might be more apparent in the dynamic experiment.

It was mentioned earlier how in the figure-8 and dynamic experiments, the mean is a bias or tracking error while the standard deviation is a variation around this because the velocities change. The standard deviation/noise in these experiments is a combination of the noise and how fast the method converges to the new velocity. Therefore, the filtered and complete versions might have higher standard deviations in the linear or angular controllers in these experiments because of the delay introduced by the moving average filter. This effect seemed to be almost negligible in the figure-8 experiment but more apparent in the dynamic experiment since the velocities changed more in the dynamic experiment.

In the simulations of the distance-based method, the filtered and camera-only versions were the best, as they had the overall lowest noise and low errors. The camera version was better than the filtered version if the velocities in the experiment were dynamic, as the filtered version performed best when the velocities were constant due to the delay introduced by the moving average filter. The LiDAR-only version was consistently the worst version, with higher errors and noise. The complete version had the lowest errors in some experiments, but since it partially used the LiDAR, it had more noise. Using the LiDAR sensor led to less shortcutting in the circle experiment, but it also led to the LiDAR version not following directly behind the leader in the constant experiment. Using the LiDAR also led to a higher position error after switching the angular velocity in the figure-8 experiment.

6.1.4 Pixel-based

In this section, the results of the pixel-based method in the simulations will be discussed. Table 5.21 shows the results for the circle experiment, table 5.22 shows the results for the constant experiment, table 5.23 shows the results for the figure-8 experiment, and table 5.24 shows the results for the dynamic experiment.

In the circle experiment, the filtered and camera versions were the better versions, as they

had the least error and noise, with very little difference between the errors. The complete version was close behind in terms of errors. The LiDAR version was the worst version, with the highest errors and noise. The LiDAR version had the least error in the m-pixel coordinates, and in the distance-based circle experiment, the LiDAR version had the least heading error. Both these parameters represent the error used in the angular control. It could be that because the LiDAR detected the leader within a cone, it detected points on the leader's sides when the robots drove in a circle. This could cause the LiDAR to detect smaller bearings/m-coordinates than the camera marker detection. It could also be that because the LiDAR was detecting the entire back of the leader, this led to it measuring lower bearing angles/m-coordinates in the circle experiment. The position error of the LiDAR version was the lowest because this version did less shortcutting than the others.

In the constant experiment, the camera and filtered versions were the best, as they had the least noise and error. They were almost identical in terms of error, and the filtered version had a little less noise than the camera-only version. The complete version had a similar error performance to the filtered and camera versions but had a little more error. The LiDAR version was the worst, as it had the most noise and error. The LiDAR version's position error in this experiment was larger than the in the other versions. This was because the LiDAR version had more distance errors than the other versions and also because the follower was not directly behind the leader. This was similar to how the LiDAR version did not follow directly behind the leader in the constant experiment for the distance-based method.

In the figure-8 experiment, the camera, filtered, and complete versions were the best and were close in terms of both overall noise and error. The complete version had the overall lowest errors, closely followed by the camera and filtered versions. The camera and filtered versions had similar noise and had the lowest noise in the linear control. Despite using the moving average filter, the complete version had the least noise in the angular control. The LiDAR version had the same issue with the figure-8 experiment as shown in figure 5.6 when it came to the position error. While the complete version also had this issue, it did not have it to the same degree. It also made up for it by having a lower distance error than the LiDAR version in this experiment, which kept the position error low.

In the dynamic experiment, the complete version was the best in terms of error, while the filtered and camera versions were the best in terms of overall noise. The filtered version had more noise in the linear control (linear velocity, n-coordinate, distance) than the camera version. This was because the linear velocity in this experiment was dynamic, and the moving average filter caused the filtered version to display the same behavior as previously discussed. It is again apparent that the moving average filter affected the dynamic experiment more than the figure-8 experiment.

Overall, the filtered and camera methods were the better versions when looking at the overall noise and errors. The complete version gave low errors in some experiments, but due to it using the LiDAR, it had more noise than the filtered and camera versions. The filtered

version struggled a little when it came to changing velocities, making the camera version the better version when the velocities were changing. The complete version struggled less in the experiments with changing velocities in the pixel-based method than in the distance-based method. Similar to the distance-based method, the versions that used the camera sensor were better than the versions using the LiDAR. Other characteristics were also similar, like the LiDAR and complete versions shortcutting less and having more position errors in the latter half of the figure-8 experiment, and the LiDAR version having more position errors in the constant experiment.

6.1.5 Comparing methods

When comparing the distance-based and pixel-based methods, the comparisons are going to rely on the velocity errors, distance/heading errors, position errors, and noise. Both methods gave good results. Overall, the pixel method had the better results here. In terms of controller characteristics, the pixel-based method did more shortcutting, which could make the distance-based controller a better choice.

The pixel-based method was the best. It had the least error and noise overall in all the experiments, except for the dynamic experiment, where it had more noise. The pixel-based method had less distance error than the distance-based method in every experiment. This could be an effect of the FoV constraints introduced in section 5.5.1 and a low maximum steady-state error for the linear control, which pushed the linear control boundaries low. It could be that if the d_{con} and d_{col} of the distance-based method were adjusted, the controller would have less distance error. The distance-based method had the least overall heading error in the constant and dynamic experiments. This could mean that the distance-based controller was better than the pixel-based controller at keeping the follower directly behind the leader. The pixel-based method shortcutting more than the distance-based method could be connected to this.

In the constant experiment, the m-coordinate error of the pixel-based LiDAR version and the heading error of the distance-based LiDAR version were relatively low despite the follower not following directly behind the leader. This might be because the LiDAR was bad at measuring angles, so not being directly behind the leader did not contribute as much to these errors.

The pixel-based method shortcuts more than the distance-based method. Comparing tables 5.17 and 5.21 shows that the pixel-based method had a larger position error despite having a much smaller distance error. Comparing the position error in tables 5.18 and 5.22, the position error for the pixel-based method was smaller, which fits with the method having less distance error. This is because the robots were driving in a line in this experiment, so no shortcutting could happen. This means that the higher position error for the pixel-based method in the circle experiment came from shortcutting. The pixel-based method also had

a higher position error in the figure-8 experiment due to shortcutting.

The pixel-based method being based on keeping a single pixel in the desired pixel position could be the reason why this method shortcuts more. Assume a square marker standing parallel with the camera, with the marker's z-axis pointing up. If the marker detection should detect a point on the marker, and the marker is rotated along the z-axis a little without any translation, the marker detection would have a hard time detecting the rotation. Rotating the marker would not give a big difference from holding the marker straight when it comes to the pixel coordinates because the pixel would still be in approximately the same coordinates as before. This could lead to the pixel-based method being worse at keeping the follower in the right position behind the leader, which could be the reason for the method shortcutting more. A way to improve upon this could be to add more than one feature point pixel and have some space between the pixels, as it would let the controller detect rotation better.

The heading error of the pixel-based method was low compared to the heading error of the distance-based method in the circle and figure-8 experiments. When the robots are driving in a path with a curved trajectory, having a larger heading error might be good. When the robots drive in a curve, and the heading error is 0 while the follower is some distance behind the leader, it means the follower is following the leader, not the leader's curved trajectory. This leads to the follower shortcutting, as previously discussed. This is why the pixel-based method had less heading and distance errors but more position errors in the circle and figure-8 experiments. Larger heading errors being good was also apparent in some of the previous experiments with curved trajectories, where a lower heading error led to larger position errors.

6.2 Physical experiments

In this section, the results of the physical experiments will be discussed, as well as the results of the sensor accuracy check for the physical sensors and experiments regarding the tuning of weights. The controllers and control versions will be compared in the same way as in section 6.1.

6.2.1 External factors that affected the physical results

In section 5.2.4, some external factors and issues related to going from the simulations to the physical system were mentioned. These included the WI-FI, lighting in the lab, and debris getting stuck to the robot wheels. These factors did not affect the experiments. In this section, the factors that affected the physical experiments and results will be mentioned.

The most apparent external factor when doing the physical experiments was the floor of the lab. In some parts of the room, the floor had long bumps. These bumps were mainly on one side of the room and covered a quarter of the room. The bumps were big enough to affect the experiments and mainly affected the pixel-based method. In addition to the bumps, the tiling in the room was not completely smooth from one tile to another. Driving over the tile changes made the robots and camera shake, which could also affect the pixel-based method. Another external factor was the USB camera. Since this camera could tilt, was bigger than the Raspberry Pi camera module, and was mounted with tape, this could also affect the physical results.

Because the bumps on the floor could make the pixel-based method have issues and become unstable, the experiments were done so that the bumps were avoided. An exception to this was the figure-8 experiment. This experiment took up so much space that driving close to or over the bumps was unavoidable without hitting the walls or going outside the motion capture's FoV. After adjustments to how the figure-8 experiment should be done, the robots drove over the bumps in the last part of this experiment, but not enough to make the pixel-based method unstable. The results of this experiment might still be affected by the bumps.

6.2.2 Characteristics of the physical sensors

The camera and LiDAR sensor's measurement accuracy and noise were checked for the physical system in the same way as in the simulated system.

Table 5.15 shows the results of the camera and LiDAR sensors measuring the distance and bearing angle. Compared to the simulated measurements, the physical camera and marker detection was a little more accurate than in the simulations. The physical camera measured closer to 0.75 meters and had less noise in the distance measurement. In the bearing measurement, it was closer to the desired value of 0 with less noise. For the physical LiDAR sensor, the distance measurement was off by a little more than 2 cm, with almost double the noise of the simulated LiDAR. It was mentioned how the physical LiDAR sensor most likely had a bias in section 5.2.6, and this was why it measured a lower distance. The bearing measurement of the physical LiDAR was also worse by almost 1 degree more compared to the simulations, with more noise. This means that the physical LiDAR was considerably worse than the simulated LiDAR, which was supposed to model the physical LiDAR of a Turtlebot.

Table 5.16 shows the results of the camera and LiDAR measuring the pixel coordinates. In the physical system, the desired pixel coordinates were $(187.2, 315.26)$. The camera measurements were very close to the desired mean values, though the measurements were noisier than in the simulations. For the physical LiDAR, the n-coordinate measurement was off by a little more than one pixel, and the m-coordinate measurement was off by a little more than

15 pixels. Since the LiDAR had a bias in the distance and was bad at finding the bearing measurements, this might have affected the measurements in pixel coordinates. This is because the distance and bearing measurements from the LiDAR were used to estimate the pixel coordinates. The LiDAR pixel measurements here also had more noise than in the simulations.

The LiDAR's distance measurement was off by a considerable amount. The LiDAR's n-coordinate measurement was off by only one pixel, despite using the distance measurement to estimate the n-coordinate. This was due to the relation between the distance and n-coordinate shown in equation (26), and that h was small. This means that a small change in the pixel coordinate would be a large change in the distance.

6.2.3 Effects of weights on the physical system

The weights of the complete version were tested in the physical setup, as was done in the simulations. Table 5.11 shows the weight test experiment for the distance-based method, and table 5.12 shows the weight test experiment for the pixel-based method.

In the distance-based method, having more weight on the camera sensor gave the least noise in most cases, while having more weight on the LiDAR never gave less noise. The overall error was mixed, as having more weight on the LiDAR gave less error in some performance metrics and having more weight on the camera gave less error in others. Because of the lower noise, more weight was put on the camera sensor.

In the pixel-based version, having more weight on the LiDAR reduced the noise considerably overall. Having the most weight on the LiDAR gave a little more error than the alternatives, but the lower noise was worth it to keep the system stable. Having the most weight on the camera barely kept the system stable, as mentioned in section 5.5.6. Having equal weights on the sensors also gave minor stability issues. It was decided that having more stability and less noise was more important than having lower errors with this version of the pixel-based method. Because of this, the weight was put on the LiDAR sensor in the experiments.

6.2.4 Distance-based

In this section, the results of the distance-based method in the physical setup will be discussed. Table 5.25 shows the results for the circle experiment, 5.26 shows the results of the constant experiment, 5.27 shows the results of the figure-8 experiment, and 5.28 shows the results of the dynamic experiment.

In the circle experiment, the camera version was the best as it had the least overall noise and errors. The complete version was close behind in both noise and error, while the filtered version was close behind in errors. The LiDAR sensor had a bias in the distance measurement; this could affect the results. It could affect the position error results, as the follower in the LiDAR version would be further back than in the other versions. This is because the LiDAR measures 0.75 meters when the follower is around 0.77 meters behind the leader. This is assuming that the bias shown in table 5.15 was consistent during the experiments. The LiDAR version had the most error in position. In the circle experiment for this method in the simulations, it had the least position error due to the LiDAR making the follower shortcut less. It could be that the LiDAR sensor made the follower shortcut less like in the simulations, but it is hard to tell, as using the LiDAR led to more position errors due to the bias.

The leader drove with more variety between the runs in the physical experiments than in the simulations. This could be because of factors such as the floor and tiles of the lab. The runs had some variety in the simulations too, but because the floor in the simulations was ideal, the leader would drive more similarly through the different runs of the experiments.

In the constant experiment, the data were only taken over 8 seconds. Because of this, the data might not be as reliable as in the other experiments where the robots drove for longer. As mentioned, there was more variety between the runs of the physical experiment. When only giving a linear velocity to the leader its trajectory in the run started curving a little after a while. The variety in curving, in addition to the short sampling time of the data, could lead to unreliable data. The curving also happened in the simulations, but it would be more similar between the runs, so its effect on the simulated experiment would be negligible.

The complete and filtered versions were the best in the constant experiment, as they had the lowest overall error and noise respectively. The filtered and LiDAR versions had high position errors. For the LiDAR version, this might be the same case as in the simulations, where the LiDAR struggled with keeping the follower directly behind the leader, which gave a higher position error. In addition, the LiDAR sensor's bias could lead to a higher position error, as it caused the follower to be further behind the leader than it should. For the filtered version, the higher position error was most likely a case of unfortunate sampling. Plot 5.7 shows that the filtered version's position error was relatively low early in the run. The data sampling was done between 35 seconds and 43 seconds. In this interval, the position error of the filtered version was rising. Unfortunate sampling might also be why the complete version had a high position error. In the plot, it started by having the least position error, but the error increased in the sampling interval. The rise in the position error of the complete and filtered versions could be due to the leader's trajectory in the runs starting to curve more during the sampling interval.

In the figure-8 experiment, there was no clear better version in terms of overall error, but the camera version was the best as it had the least noise. The position error in the LiDAR

version was larger than in the rest of the versions. This was for the same reason as discussed in the simulations and shown in plot 5.6. The LiDAR struggled to keep its position after the angular velocity in the experiment had changed. Some of this position error could also be because of the LiDAR sensor's bias.

In the dynamic experiment, the camera version was the best as it had the least error with low noise. The complete version was close behind in overall noise and error. The camera version had the least noise in the linear control. Because of the moving average filter, both the complete and filtered versions had considerable noise in the linear control. They did, however, have low noise in the angular control, with the complete version having the least of all versions.

Overall, either the camera or complete version would be the best in the physical distance-based method. These were the versions with the least noise and errors. The filtered version had less noise in the constant experiment, but this experiment was said to be less reliable than the others. The LiDAR version was the worst of the versions, with high noise and errors. The LiDAR version also had a bias that affected the results. The physical LiDAR version had the same issues as discussed in the simulations, like the higher position error in the latter half of the figure-8 experiment and not following directly behind the leader in the constant experiment.

6.2.5 Pixel-based

In this section, the results of the pixel-based method in the physical setup will be discussed. Table 5.29 shows the results for the circle experiment, 5.30 shows the results for the constant experiment, 5.31 shows the results for the figure-8 experiment, and 5.32 shows the results for the dynamic experiment.

In section 6.2.1, it was mentioned that bumps and tile changes in the floor could affect the pixel-based method. When the follower drove over bumps, the camera was lifted a little. In section 5.2.3, it was mentioned how small changes in the tilt of the camera could affect the n-coordinate measurements in the pixel-based method. When the camera got lifted by driving over the bumps, it caused the pixel-based method to detect higher n-coordinates than it should. Due to this, the follower would slow down, almost stopping when the follower drove over bumps. The leader would keep driving, causing the distance between the robots to grow bigger. In the worst case, this would cause the system to become unstable due to the spike in n-coordinate errors if the follower could not catch up to the leader.

The robots driving over tile changes could similarly affect the follower but to a lesser degree. The follower would slow down when driving over tile changes, but it would not cause instability. Driving over tile changes caused the camera and robots to shake slightly. The shaking mainly affected the detection of pixels in the n-coordinates, as they were sensitive to

the camera shaking slightly up and down when the robot drove over tile changes. Due to the camera shaking, the controller measured n-coordinates further up or down in the image for a moment, which led to wrong measurements. This made the follower think that it was closer to or further away from the leader than it was, causing the follower to slow down or speed up. Since the camera was shaking when the follower was driving, and the shaking affected the n-coordinates, the linear control of the pixel-based method contained more noise. Since the USB camera was big and mounted with tape, it could be that using a Raspberry Pi camera module would lead to less shaking, as it had a proper mount and was more compact.

It was mentioned how the FoV constraints of the pixel-based method were changed from what was used in (Miao et al., 2021), as the old constraints made the follower drive backward initially. The new constraints, especially for the n-coordinate, were smaller. It could be that this affected the stability of the system when driving over bumps. The smaller constraints gave smaller boundaries that the error had to pass before the system became unstable. Since the boundaries also depend on the maximum steady-state error, it is not certain if the new FoV constraints affected the stability when driving over bumps. The steady-state errors could not be optimized in the physical pixel-based method due to noise and instability, so they were quite high, making it so that the boundaries were not pushed low because of them.

In the circle experiment, there are only results from the camera, LiDAR, and complete versions. This is because the filtered version could not be kept stable for 300 seconds. The camera shaking combined with the moving average filter caused the follower to become unstable in the filtered version. The moving average filter was tuned down to the point where it barely filtered anything to verify that it was the filter that caused the issue. Without most of the filtering, the system was stable.

The complete version was the best in the circle experiment, as it had the lowest overall errors and the least noise in the linear control. The camera version had the least noise in the angular control but most in the linear control. The complete version had more weight on the LiDAR sensor in the physical pixel-based method, as previously mentioned. The noise results can be explained by the camera shaking and the LiDAR being noisy in angle measurements. Since the complete version put more weight on the LiDAR, it and the LiDAR version had more noise in the angular control part. Due to camera shaking, the camera version had more noise in the linear control. The LiDAR version had the most position error. This could be due to the large measurement error the LiDAR had in the m-coordinates, as shown in table 5.16. The complete version had the least position error despite putting the most weight on the LiDAR. This could be because the complete version used a combination of the camera and LiDAR measurements, not only the LiDAR measurements.

An important thing to note is that the complete version of the pixel-based method used the camera to measure the distance and bearing angle. As seen in table 5.15, there was a difference between what the camera and LiDAR would measure the distance and bearing as. This could affect the distance and heading error results of the complete version since

this version put the weight on the LiDAR measurements but used the camera to measure distance and bearing.

In the constant experiment, the data was only sampled for 8 seconds. Because of this, the data might be less reliable than in the other experiments, as was discussed for the same experiment for the physical distance-based method. The complete version had the least noise overall, with the least error in some metrics, making it the best version. In terms of error, there was no version that was better. However, the versions using the camera struggled more with noise and errors in the linear control while the versions using the LiDAR struggled more with angular control. The filtered version had the least error in the angular velocity and m-coordinate error. The LiDAR version had the least velocity and n-coordinate error, and the complete version had the least distance and heading error.

An example of how the pixel-based method slowed down due to camera shaking is shown in figure 5.8. Around the 20-second mark, the versions using the camera slowed down heavily. There were also instances where the follower slowed down a little less, as indicated by the smaller spikes in the linear velocity. If the cause of the follower slowing down had been random, the spikes indicating the breaking would have been more random in time between the different versions. Since the spikes were consistent in time between the different runs, this could mean that the robots drove over something like changes in tiles. Driving over tile changes caused camera shaking, which could cause the follower to slow down, as previously mentioned. The complete version was affected the least out of the versions using the camera since it relied less on the camera. The overall velocity of the versions that used the camera was higher than in the LiDAR version. There were several instances where the camera and filtered versions reached the maximum velocity of the Turtlebot, as indicated by some of the tops of the linear velocity being cut in the plot. Camera shaking might be the reason for this, with the follower thinking it was further away from the leader than it was and speeding up, or the follower having to catch up to the leader after slowing down.

In the figure-8 experiment, the complete version was the best, as it had low errors and the least noise in the linear control. The camera version had the least noise overall in the angular control but more overall errors. The noise results could again be a consequence of the camera shaking and LiDAR being bad with angles. From figure 5.9, there were spikes in the position error of the filtered and camera versions after switching velocity. These were likely caused by the bumps on the floor. As mentioned in section 6.2.1, most experiments were set up to avoid bumps, but in the figure-8 experiments, the robots had to drive over them. Even though this did not make the versions that used the camera unstable, it could make the follower stop or slow down for some time due to it thinking it was too close to the leader. The filtered and camera versions also had higher overall position errors after the switch. This came from them shortcutting more after switching velocity as they were trying to catch up to the leader due to slowing down on the bumps. The figure also shows that the LiDAR version had a high overall position error. This was most likely because of the same reason as discussed in the circle experiment.

In the dynamic experiment, the complete version was the best, as it had the lowest errors overall, with low noise. The camera version had the least noise in the angular control. The LiDAR version had the least noise in the linear control, with the complete version close behind. The noise was like this because of the camera shaking and the LiDAR being bad with angles, as was the case in the other experiments. The reason why the complete version had a little more noise than the LiDAR version was most likely because of the moving average filter's delay.

The best version of the physical pixel-based method would be either the complete or the LiDAR versions. The filtered version was not good enough, as it failed one of the experiments. The camera and filtered versions had low noise and errors when it came to the angular control, but they were very unstable with much noise when it came to the linear control. The camera and filtered versions had notable issues, where camera tilt and bumps/tile changes on the floor affected these versions of the pixel-based method because they were sensitive to small disturbances to the camera. The versions using the camera were therefore not very robust. Because of this, using the LiDAR was the better option. While using the LiDAR was worse when it came to angles, it was more robust than using the camera. The LiDAR had issues with measurements, as shown in the measurement tables 5.15 and 5.16, but using the LiDAR made the physical pixel-based method more robust and stable.

6.2.6 Comparing methods

The distance-based method had the least distance error in all the experiments, except for the dynamic experiment. The reason why the pixel-based method had more distance error than the distance-based method in the physical setup when it had less in the simulations is most likely because the maximum steady-state errors of the pixel-based method could not be optimized in the physical setup. This led to the distance in the pixel-based method being higher than in the distance-based method.

In terms of error, the physical distance-based method was better at linear control, while the physical pixel-based method was better at angular control. The simulated pixel-based method had lower overall errors than the simulated distance-based method, so why did the physical pixel-based method not have lower overall errors than the physical distance-based method? This was because, in the simulations, the versions of the pixel-based method that used the camera were the best overall. In the physical pixel-based method, the versions using the camera struggled with linear control because of the camera shaking and floor bumps. The distance-based method was more robust towards the shaking or the camera being lifted/lowered a little, as it did not use the image from the camera directly. In addition, the LiDAR sensor struggled in the physical setup, making the performance of all the versions that used the LiDAR worse. This made it so that the distance-based versions that used the camera gave better overall results in the error, as they were not affected by issues the pixel-

based camera versions and LiDAR sensor.

Overall, the pixel-based method had more noise in the linear control than the distance-based method. This is again because the physical pixel-based versions that used the camera struggled with camera shaking, making it so that there was more noise in the linear control. The pixel-based method had lower noise in the angular control. The pixel-based method had more position errors than the distance-based method. This was because the pixel-based method had a larger distance error due to bad optimization. In the circle and figure-8 experiments, the higher position error also came from the pixel-based method shortcutting more than the distance-based method, as discussed in section 6.1.5. This matches with the pixel-based method having less heading error in these experiments than the distance-based method.

The pixel-based method struggled with bumps on the floor and camera shaking to the point where it made the system unstable or very noisy. This made the camera and filtered versions of the pixel-based methods the worst in the physical experiments. Adding a LiDAR improved the stability of the pixel-based method. The complete and LiDAR versions of the pixel-based method were more stable than the versions using the camera. The issue with solely relying on the LiDAR is that it is not good at handling angles. When using the LiDAR, the pixel-based method became better in the linear control. In exchange, it became worse in the angular control in most cases. In the distance-based method, using only the LiDAR gave more noise and worse results than using the camera. The overall best method in the physical setup would be the distance-based method with either the camera version or the complete version, as these were the versions with the lowest errors and noise. The pixel-based method with the complete version did well too, but it relied more on the LiDAR than the camera, which had a bias and was bad at measuring angles.

6.3 Comparing results to other papers

In this section, the results from the experiments will be compared to the results in (Verginis et al., 2015) and (Miao et al., 2021). Since the only sensor the papers use is the camera, only the camera version of the controllers will be compared to their results.

In (Verginis et al., 2015), only simulations were done. They used different parameter values for the controller than what was used in this thesis. They had smaller maximum steady-state errors, $\rho_{d,\infty} = 0.0625$ and $\rho_{\beta,\infty} = 1.15$. They also made the system converge faster by setting $l = 0.6$, had different control gains, and used more than one follower. These steady-state errors were not chosen in this thesis, as they could not keep the system stable, and making the system converge this fast also caused issues with stability.

Their experiment was different, as they drove their robots with a curved trajectory con-

sisting of two turns, as shown in Fig.3 in their paper. Because of this, it is hard to compare their simulation results to the ones for the simulated distance-based method in this thesis. What can be seen from Fig. 4 and 5 in their paper is that their errors converge very close to 0. This is most likely due to their low maximum steady-state errors. The errors in this thesis did not converge to values this low because of larger steady-state errors, even though the errors were quite low.

In (Miao et al., 2021), they did both simulations and physical experiments. It was mentioned how the FoV constraints used in this thesis were different from the ones used in their paper. Since their paper presents a formation controller, their desired position was not directly behind the leader but behind and a little to the side. Other values, such as the control gain, desired transient time constant, maximum steady-state errors, etc. were also different.

In the simulations, they used maximum steady-state errors of 10 and a desired transient time constant of $l = 0.6$. In this thesis, $\rho_{n,\infty} = 20$, $\rho_{m,\infty} = 30$, and $l = 0.1$, for the simulated pixel-based method, due to stability. Figure 2. in their paper shows their simulation results for an experiment where the robots drove in a straight line with constant velocity. The noise levels between their simulations and the ones in this thesis for the camera version seem similar. The difference is that while the simulations in this thesis had very little noise, the simulations in (Miao et al., 2021) and (Verginis et al., 2015) did not seem to contain any noise at all. It could be that they did not introduce noise to their simulations, while the simulations in this thesis had noise from the ROS packages. The lack of noise in their simulations could be why they could choose smaller steady-state errors and make the system converge faster. The errors in the simulations in (Miao et al., 2021) seem to converge very close to 0. The pixel coordinate errors in this thesis for the constant experiment were also low. Theirs most likely converge to a smaller error due to the smaller steady-state errors.

Their physical results for the static gain controller are shown in Fig. 4 and 5. In their physical experiment, the robots drove in a circle, $l = 0.1$, and their maximum steady-state errors were set to 20. In the physical pixel-based method in this thesis, $l = 0.1$, $\rho_{n,\infty} = 30$, and $\rho_{m,\infty} = 60$. Their physical system converged to a 7-pixel error in the n-coordinate and an 18-pixel error in the m-coordinate. In this thesis, the n-coordinate and m-coordinate errors converged to around 6.5 and 31 pixels respectively for the circle experiment with the pixel-based camera version. The larger m-coordinate error was most likely due to the large steady-state error $\rho_{m,\infty}$, which could not be pushed lower due to stability issues. The lower n-coordinate error, despite a larger steady-state error, could come from the FoV constraints in this thesis being smaller, especially for the n-coordinate, which pushed the boundaries lower.

Fig. 4 and 5 in (Miao et al., 2021) show that their physical system had a considerable amount of noise. While their system had less noise than the camera version in this thesis, it looks like it had the same issues with the follower slowing down. As previously discussed, this was likely due to the camera shaking while the follower was driving. An example of

this was shown in figure 5.8 in this thesis. Between the 20 and 40 second mark in Fig.4 in their paper, the follower was slowing down. This also seems to have caused their angular velocity to be noisy. In this thesis, the camera shaking was mitigated by adding the LiDAR and was somewhat reduced by having the filter. If the physical pixel-based versions that used the cameras in this thesis were excluded, the maximum steady-state errors could likely be pushed lower to get better performance. In their paper, the static gain controller drove forward instantly instead of backing up like it did in this thesis. This was most likely because of their camera and tracking object being on poles of different sized, which gave a larger h. The follower backing up initially was the reason why the FoV constraints in this thesis had to be changed.

Chapter 7

Conclusion

In this thesis, two methods for platooning control without inter-vehicle communication were simulated and implemented with two Turtlebot3 Waffle-Pi robots. One method was based on finding the distance and bearing angle between the leader and follower. The second method was based on keeping a feature point in the desired pixel coordinates. ROS, ROS packages, and Gazebo were used to simulate and implement the controllers. Both controllers were prescribed performance controllers, and they only used visual sensor information. The controllers were evaluated by doing experiments where the robots drove in different patterns.

Additions to the original control methods were made in an attempt to make the methods more robust and less noisy. Adding a moving average filter reduced noise in some cases, but caused issues in others due to the delay introduced by the filter. Using a LiDAR instead of a camera made the controller more robust in cases where visual information was noisy or lost. In cases where the camera was reliable, it had less noise than the LiDAR. The LiDAR struggled with measuring angles, and in the physical setup, the LiDAR had a bias. A version where the camera, LiDAR, and filter were combined was also used.

Both controllers worked well in the simulations, with the pixel-based method having less error and noise. The pixel-based method struggled in the physical setup. It was sensitive to the camera shaking when the follower drove over tile changes and uneven flooring. This would cause noise or, in the worst case, instability. The method was also sensitive to camera tilt. The tilt of the camera had to be adjusted before every physical experiment. Most of these issues were reduced by making use of the LiDAR instead of the camera, and using the complete version made this method more robust. The distance-based method was better in the physical setup and easier to use than the pixel-based method due to the aforementioned reasons. In addition, if the path the robots took was curved, the pixel-based method would shortcut more.

In the simulations, the camera and filtered versions had the least noise and error in both

control methods. The camera and marker detection had very little noise in the simulations, making it so that these versions gave better, less noisy results. In the physical setup, the camera and complete versions gave the least noise and error for the distance-based method; the weight was on the camera in the complete version. Since the pixel-based method struggled with using the camera in the physical experiments, the LiDAR version and complete version with the weight on the LiDAR were better for the physical pixel-based method.

7.1 Future Works

Having adaptive control gain could be useful. In this thesis, the control gains in the controllers were tuned and found through testing to fit each specific pattern that the robots drove in. This makes it difficult to use the robots in real-world applications, as the gains were manually adjusted through testing and a priori knowledge of the trajectory and velocities of the leader. Generalized gains that would work for more patterns could be found through testing. A better solution would be adaptive gain, which would make the controllers more generalized and better at handling a larger variety of patterns.

Better sensors could be used to perform the experiments in the future. A camera that is smaller and mounted better on the follower could lead to less shaking and issues for the pixel-based method. Finding a way to make the Raspberry Pi camera module work could help. Using a different LiDAR with better angular resolution could eliminate the bias that the LiDAR in this thesis had, in addition to making the versions that used the LiDAR better at angular control. Adding a constant to the LiDAR measurements could also help with the bias. Adding more feature points to the pixel-based method could make it more robust and shortcut less.

Both control methods shortcut, with the pixel-based method shortcutting more than the distance-based. Ways to make the controllers shortcut less could be explored. For example, adding a tolerance for the angular controllers or making the follower take the trajectory of the leader into account instead of only following the leader directly.

References

Abualhoul, M. Y., Marouf, M., Shagdar, O., & Nashashibi, F. (2013). Platooning control using visible light communications: A feasibility study. In *16th international ieee conference on intelligent transportation systems (itsc 2013)* (p. 1535-1540). doi: 10.1109/ITSC.2013.6728448

ACEA. (n.d.-a). *What are the benefits of truck platooning?*

ACEA. (n.d.-b). *What is truck platooning?*

Acea. (2016). *What is the european truck platooning challenge?* `https://www.acea.auto/fact/what-is-the-european-truck-platooning-challenge/`. ([Online; accessed 26-August-2021])

Ali, A., Garcia, G., & Martinet, P. (2015). Urban platooning using a flatbed tow truck model. In *2015 ieee intelligent vehicles symposium (iv)* (p. 374-379). doi: 10.1109/IVS.2015.7225714

Baardseth, E. V. B. (n.d.). *Vehicle detection and pose estimation in autonomous convoys.*

Bechlioulis, C., & Rovithakis, G. (2008, 11). Robust adaptive control of feedback linearizable mimo nonlinear systems with prescribed performance. *Automatic Control, IEEE Transactions on*, *53*, 2090 - 2099. doi: 10.1109/TAC.2008.929402

Bechlioulis, C. P., Heshmati-alamdari, S., Karras, G. C., & Kyriakopoulos, K. J. (2019). Robust image-based visual servoing with prescribed performance under field of view constraints. *IEEE Transactions on Robotics*, *35*(4), 1063-1070. doi: 10.1109/TRO.2019.2914333

Bechlioulis, C. P., & Rovithakis, G. A. (2014). A low-complexity global approximation-free control scheme with prescribed performance for unknown pure feedback systems. *Automatica*, *50*(4), 1217-1226. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0005109814000582` doi: https://doi.org/10.1016/j.automatica.2014.02.020

Belkhouche, F., & Belkhouche, B. (2005). Modeling and controlling a robotic convoy using guidance laws strategies. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, *35*(4), 813-825. doi: 10.1109/TSMCB.2005.846646

Benhimane, S., Malis, E., Rives, P., & Azinheira, J. (2005). Vision-based control for car platooning using homography decomposition. In *Proceedings of the 2005 ieee international conference on robotics and automation* (p. 2161-2166). doi: 10.1109/ROBOT.2005.1570433

Bergenhem, C. (2015). Approaches for facilities layer protocols for platooning. In *2015 ieee 18th international conference on intelligent transportation systems* (p. 1989-1994). doi: 10.1109/ITSC.2015.322

Bergenhem, C., Shladover, S., Coelingh, E., Englund, C., & Tsugawa, S. (2012). Overview of platooning systems..

Collins, R. (2007). *Lecture 12: Camera projection.* http://www.cse.psu.edu/ rtc12/CSE486/lecture12.pdf. ([Online; accessed 22-December-2021])

Daviet, P., & Parent, M. (1996). Longitudinal and lateral servoing of vehicles in a platoon. In *Proceedings of conference on intelligent vehicles* (p. 41-46). doi: 10.1109/IVS.1996.566349

Dekra. (2018). *Platooning projects around the world.* `https://www.dekra-roadsafety.com/en/platooning-projects-around-the-world/`. ([Online; accessed 25-August-2021])

Detection of aruco markers. (n.d.). `https://docs.opencv.org/3.4/dc/dbb/tutorial_py_calibration.html`. (Accessed: 2022-03-16)

Dudek, G., & Jenkin, M. (2010). *Computational principles of mobile robotics.* Cambridge University Press.

Elrob. (n.d.). *Elrob 2016 team information.* `https://www.elrob.org/files/elrob2016/TeamInformation_MuCAR.pdf`. ([Online; accessed 11-May-2022])

Englund, C., Chen, L., Ploeg, J., Semsar-Kazerooni, E., Voronov, A., Bengtsson, H. H., & Didoff, J. (2016). The grand cooperative driving challenge 2016: boosting the introduction of cooperative automated vehicles. *IEEE Wireless Communications*, *23*(4), 146-152. doi: 10.1109/MWC.2016.7553038

Farag, A., Mahfouz, D. M., Shehata, O. M., & Morgan, E. I. (2019). A novel ros-based joining and leaving protocols for platoon management. In *2019 ieee international conference on vehicular electronics and safety (icves)* (p. 1-6). doi: 10.1109/ICVES.2019.8906393

Feng, S., Zhang, Y., Li, S. E., Cao, Z., Liu, H. X., & Li, L. (2019). String stability for vehicular platoon control: Definitions and analysis methods. *Annual Reviews in Control*, *47*, 81-97. Retrieved from `https://www.sciencedirect.com/science/article/pii/S1367578819300240` doi: https://doi.org/10.1016/j.arcontrol.2019.03.001

Fernandez, L., Avila, V., & Gonçalves, L. M. G. (2017). A generic approach for error estimation of depth data from (stereo and rgb-d) 3d sensors..

Fries, C., Burger, P., Kallwies, J., Naujoks, B., Luettel, T., & Wuensche, H.-J. (2017). How mucar won the convoy scenario at elrob 2016. In *2017 ieee 20th international conference on intelligent transportation systems (itsc)* (p. 1-7). doi: 10.1109/ITSC.2017.8317711

Fries, C., Luettel, T., & Wuensche, H.-J. (2013). Combining model- and template-based vehicle tracking for autonomous convoy driving. In *2013 ieee intelligent vehicles symposium (iv)* (p. 1022-1027). doi: 10.1109/IVS.2013.6629600

Fries, C., & Wuensche, H.-J. (2014). Monocular template-based vehicle tracking for autonomous convoy driving. In *2014 ieee/rsj international conference on intelligent robots and systems* (p. 2727-2732). doi: 10.1109/IROS.2014.6942935

Fries, C., & Wuensche, H.-J. (2015). Autonomous convoy driving by night: The vehicle tracking system. In *2015 ieee international conference on technologies for practical robot applications (tepra)* (p. 1-6). doi: 10.1109/TePRA.2015.7219675

Fullbay. (n.d.). *4 things you need to know about truck platooning.*

Gallagher, J. (n.d.). *Exclusive: Feds award major truck platooning contract to california path.*

Garrido-Jurado, S., Muñoz-Salinas, R., Madrid-Cuevas, F., & Marín-Jiménez, M. (2014, 06). Automatic generation and detection of highly reliable fiducial markers under occlusion. *Pattern Recognition*, *47*, 2280–2292. doi: 10.1016/j.patcog.2014.01.005

Holtet, J. A. (2018). *Lidar.* https://snl.no/lidar. ([Online; accessed 22-December-2021])

Hung, P. D., Vinh, T. Q., & Dung, N. T. (2012). A simplified control for robot convoy. In *2012 international conference on control, automation and information sciences (iccais)* (p. 288-293). doi: 10.1109/ICCAIS.2012.6466604

ITS Asia-Pacific. (n.d.). *2008-2012 energy its (japan).*

Jo, Y., Kim, J., Oh, C., Kim, I., & Lee, G. (2019). Benefits of travel time savings by truck platooning in korean freeway networks. *Transport Policy*, *83*, 37-45. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0967070X19300447` doi: https://doi.org/10.1016/j.tranpol.2019.09.003

Kim, T.-W., Jang, W.-S., Jang, J., & Kim, J.-C. (2020). Camera and radar-based perception system for truck platooning. In *2020 20th international conference on control, automation and systems (iccas)* (p. 950-955). doi: 10.23919/ICCAS50221.2020.9268196

Koenig, N., & Howard, A. (2004). Design and use paradigms for gazebo, an open-source multi-robot simulator. In *2004 ieee/rsj international conference on intelligent robots and systems (iros) (ieee cat. no.04ch37566)* (Vol. 3, p. 2149-2154 vol.3). doi: 10.1109/IROS.2004.1389727

Kwon, J.-W., & Chwa, D. (2014). Adaptive bidirectional platoon control using a coupled sliding mode control method. *IEEE Transactions on Intelligent Transportation Systems*, *15*(5), 2040-2048. doi: 10.1109/TITS.2014.2308535

Lin, J., Miao, Z., Zhong, H., Peng, W., Wang, Y., & Fierro, R. (2021). Adaptive image-based leader–follower formation control of mobile robots with visibility constraints. *IEEE Transactions on Industrial Electronics*, *68*(7), 6010-6019. doi: 10.1109/TIE.2020.2994861

Manz, M., Luettel, T., von Hundelshausen, F., & Wuensche, H.-J. (2011). Monocular model-based 3d vehicle tracking for autonomous vehicles in unstructured environment. In *2011 ieee international conference on robotics and automation* (p. 2465-2471). doi: 10.1109/ICRA.2011.5979581

Miao, Z., Zhong, H., Lin, J., Wang, Y., Chen, Y., & Fierro, R. (2021). Vision-based formation control of mobile robots with fov constraints and unknown feature depth. *IEEE Transactions on Control Systems Technology*, *29*(5), 2231-2238. doi: 10.1109/TCST.2020.3023415

M'Sirdi, N. K. (2021). Autonomous vehicle platooning and motion control. In B. Hajji, A. Mellit, G. Marco Tina, A. Rabhi, J. Launay, & S. E. Naimi (Eds.), *Proceedings*

of the 2nd international conference on electronic engineering and renewable energy systems (pp. 3–19). Singapore: Springer Singapore.

Muratori, M., Holden, J., Lammert, M., Duran, A., Young, S., & Gonder, J. (2017, 03). Potential for platooning in u.s. highway freight transport. *SAE International Journal of Commercial Vehicles*, *10*. doi: 10.4271/2017-01-0086

Mutz, F., Cardoso, V., Teixeira, T., Jesus, L. F. R., Golçalves, M. A., Guidolini, R., ... De Souza, A. F. (2017). Following the leader using a tracking system based on pre-trained deep neural networks. In *2017 international joint conference on neural networks (ijcnn)* (p. 4332-4339). doi: 10.1109/IJCNN.2017.7966404

Nacer, K., Dahmani, A., & Nasser, H. (2020). Overview on modeling for control of autonomous road vehicles platoon.

News, G. C. C. (n.d.). *Driver shortage is pan-european.*

Nymoen, K. (2013). *Methods and technologies for analysing links between musical sound and body motion* (Unpublished doctoral dissertation). University of Oslo. (Chapter 3)

Open Robotics. (n.d.). *Ros.* `https://www.ros.org/`. ([Online; accessed 9-May-2022])

OpenCV. (n.d.). *Detection of aruco markers.* `https://docs.opencv.org/4.x/d5/dae/tutorial_aruco_detection.html`. ([Online; accessed 16-March-2022])

Ortega, R., Astolfi, A., & Barabanov, N. E. (2002). Nonlinear pi control of uncertain systems: an alternative to parameter adaptation. *Systems and Control Letters*, *47*(3), 259-278. Retrieved from `https://www.sciencedirect.com/science/article/pii/S0167691102002128` doi: https://doi.org/10.1016/S0167-6911(02)00212-8

PAL Robotics. (n.d.). *Gazebo ros link attacher.* `https://github.com/pal-robotics/gazebo_ros_link_attacher`. ([Online; accessed 9-May-2022])

Park, M.-S., & Chwa, D. (2009). Orbital stabilization of inverted-pendulum systems via coupled sliding-mode control. *IEEE Transactions on Industrial Electronics*, *56*(9), 3556-3570. doi: 10.1109/TIE.2009.2021178

Petrov, P. (2008). A mathematical model for control of an autonomous vehicle convoy. *Lateral (steering)*, *5*, 8.

Pnorental. (n.d.). *Truck platooning: The future of road transport?*

Rezgui, J., Gagné, É., Blain, G., St-Pierre, O., & Harvey, M. (2020). Platooning of autonomous vehicles with artificial intelligence v2i communications and navigation algorithm. In *2020 global information infrastructure and networking symposium (giis)* (p. 1-6). doi: 10.1109/GIIS50753.2020.9248490

Rhody, H. (2015, October). *Lecture 10: Hough circle transform.* Rochester Institute of Technology.

Road Safety GB. (2017). *'platooning' lorry trial announcement: stakeholder reaction.* `https://roadsafetygb.org.uk/news/platooning-lorry-trial-announcement-stakeholder-reaction-5963/`. ([Online; accessed 24-August-2021])

Roberts, J. (2016). *Success of truck platooning challenge clears way for real-life convoys – steve phillips, cedr.* `https://ec.europa.eu/research-and-innovation/en/horizon-magazine/success-truck-platooning-challenge-clears-way-real-life-convoys-steve-phillips-cedr`. ([Online; accessed 26-August-2021])

Ros wiki. (n.d.). http://wiki.ros.org/. ([Online; accessed 9-May-2022])

Saracco, R. (2019). *Platooning does not seem a viable business.* https://cmte.ieee.org/futuredirections/2019/01/16/platooning-does-not-seem-a-viable-business/. ([Online; accessed 24-August-2021])

Scheuer, A., Simonin, O., & Charpillet, F. (2009). Safe longitudinal platoons of vehicles without communication. In *2009 ieee international conference on robotics and automation* (p. 70-75). doi: 10.1109/ROBOT.2009.5152629

Schneider, F. E. (Unknown). *Elrob – the european land robot trial.* https://www.elrob.org/. ([Online; accessed 26-August-2021])

Science Reference Section, Library of Congress. (2019). *Why do geese fly in a v?* https://www.loc.gov/everyday-mysteries/zoology/item/why-do-geese-fly-in-a-v/. ([Online; accessed 29-August-2021])

Sheikholeslam, S., & Desoer, C. (1993). Longitudinal control of a platoon of vehicles with no communication of lead vehicle information: a system level study. *IEEE Transactions on Vehicular Technology*, *42*(4), 546-554. doi: 10.1109/25.260756

Smith, S. W. (1999). *The scientist and engineer's guide to digital signal processing*. California Technical Publishing.

Stankovic, S., Stanojevic, M., & Siljak, D. (2000). Decentralized overlapping control of a platoon of vehicles. *IEEE Transactions on Control Systems Technology*, *8*(5), 816-832. doi: 10.1109/87.865854

Swaroop, D., & Rajagopal, K. (2001). A review of constant time headway policy for automatic vehicle following. In *Itsc 2001. 2001 ieee intelligent transportation systems. proceedings (cat. no.01th8585)* (p. 65-69). doi: 10.1109/ITSC.2001.948631

Torabi, S., & Wahde, M. (2018). Fuel-efficient driving strategies for heavy-duty vehicles: A platooning approach based on speed profile optimization. *Journal of Advanced Transportation*, *2018*, 1-12.

TRIMIS. (2021). *Konvoi.* https://trimis.ec.europa.eu/project/konvoi. ([Online; accessed 26-August-2021])

Turck, J. D. (n.d.). *The european truck platooning challenge: Transforming europe's road transport on autopilot.*

UC Berkeley. (n.d.). *Our mission.*

Ucar, S., Ergen, S. C., & Ozkasap, O. (2018). Ieee 802.11p and visible light hybrid communication based secure autonomous platoon. *IEEE Transactions on Vehicular Technology*, *67*(9), 8667-8681. doi: 10.1109/TVT.2018.2840846

The unicycle model. (n.d.). http://faculty.salina.k-state.edu/tim/robotics_sg/Control/kinematics/unicycle.html. (Accessed: 2022-03-16)

University of Idaho. (n.d.). *Shock waves.*

Verginis, C. K., Bechlioulis, C. P., Dimarogonas, D. V., & Kyriakopoulos, K. J. (2015). Decentralized 2-d control of vehicular platoons under limited visual feedback. In *2015 ieee/rsj international conference on intelligent robots and systems (iros)* (p. 3566-3571). doi: 10.1109/IROS.2015.7353875

Vinel, A., Lyamin, N., & Isachenkov, P. (2018). Modeling of v2v communications for c-its

safety applications: A cps perspective. *IEEE Communications Letters*, *22*(8), 1600-1603. doi: 10.1109/LCOMM.2018.2835484

WBOC. (n.d.). *Global truck platooning systems market.*

Wei, C., Chen, Q., Liu, J., & Yin, Z. (2020, 08). An overview of prescribed performance control and its application to spacecraft attitude system. *Proceedings of the Institution of Mechanical Engineers, Part I: Journal of Systems and Control Engineering*, *235*, 095965182095255. doi: 10.1177/0959651820952552

Yang, Q., Liu, S., & Jiang, X. (2018). An autonomous multi-vehicles queue following control system with vision and laser and simulation on ros. In *2018 ieee 8th annual international conference on cyber technology in automation, control, and intelligent systems (cyber)* (p. 747-752). doi: 10.1109/CYBER.2018.8688198

Yazbeck, J., Scheuer, A., Simonin, O., & Charpillet, F. (2011). Improving near-to-near lateral control of platoons without communication. In *2011 ieee/rsj international conference on intelligent robots and systems* (p. 4103-4108). doi: 10.1109/IROS.2011.6094929

Zhao, X., Yao, W., Li, N., & Wang, Y. (2017). Design of leader's path following system for multi-vehicle autonomous convoy. In *2017 ieee international conference on unmanned systems (icus)* (p. 132-138). doi: 10.1109/ICUS.2017.8278329