CHALMERS
UNIVERSITY OF TECHNOLOGY

Path following for ATRs using embedded Nonlinear Model Predictive Control

A path following control system with localization based on photogrammetry and odometry

Master's thesis in Systems, control and mechatronics

LUCAS JUTVIK
ADAM KHALIF

DEPARTMENT OF ELECTRICAL ENGINEERING
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2021
www.chalmers.se

Master's thesis 2021

Path following for ATRs using embedded Nonlinear Model Predictive Control

A path following control system with localization based on photogrammetry and odometry

LUCAS JUTVIK

ADAM KHALIF

Department of Electrical engineering
Chalmers University of Technology
Gothenburg, Sweden 2021

Path following for ATRs using embedded Nonlinear Model Predictive Control
A path following control system with localization based on photogrammetry and odometry
ADAM KHALIF, LUCAS JUTVIK

Supervisor: Per-Lage Götvall, Volvo GTO
Examiner: Emmanuel Dean, Department of Electrical Engineering

Master's Thesis 2021
Department of Electrical engineering
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Gothenburg, Sweden 2021

Path following for ATRs using embedded Nonlinear Model Predictive Control
A path following control system with localization based on photogrammetry and odometry
LUCAS JUTVIK
ADAM KHALIF
Department of Electrical engineering
Chalmers University of Technology

Abstract

A path-following controller is essential for an Autonomous Transport Robot. Control systems based on Model Predictive Control have recently become more popular. This rise in popularity is partly because of new algorithms, which have reduced the computational cost for solving MPC problems. This thesis investigates the current technical feasibility of using an embedded two-step control scheme for path-following. The first step is an MPC that controls the velocity of the robot, while the second step of the scheme is a PID controller that controls the actuators of the robot's wheels. One advantage of this setup is that no detailed model of the actuators is needed. Odometry from wheel encoders and photogrammetry from cameras mounted on the ceiling were both utilized for localization. Sensor fusion was performed with the look-and-move method to counteract latency in the photogrammetry. The odometry was used while moving and the accumulated errors were removed using the photogrammetry when standing still. The performance of the localization system was evaluated both by fusing the odometry and photogrammetry sensor inputs as well as testing them individually. While following a reference trajectory, the MPC solve time averaged less than 10 ms, enabling a control frequency of more than 100 Hz. The look-and-move method achieved high accuracy and managed to correct the odometry drift, indicating that the embedded path-following system may be a feasible solution.

Keywords: path following, localization, autonomous transport robot, NMPC, photogrammetry, odometry, ROS2, PANOC, sensor fusion.

Acknowledgements

We would like to extend our sincere thanks to Emmanuel Dean for all the invaluable input he provided as our supervisor and for all the laughs shared. Secondly, we would like to express our gratitude to Knut Åkesson for lending his experience throughout this project. We would also like to extend a special thanks to Per-Lage Götvall for providing assistance wherever needed and without whom this would not have been possible. Finally, we would like to thank our three sister projects, it has been fun!

Adam Khalif & Lucas Jutvik, Gothenburg, June 2021

Contents

List of Figures

List of Tables

1
Introduction

1.1 Background

The current production line in the Volvo factories has the material needed at each workstation provided in assembly kits. Storage space close to the worker is very scarce and materials are therefore placed in storage racks away from the station while still being in close vicinity. The more variance each product has regarding its components the bigger the need for storage space, which is currently hard to provide. This, in turn, either limit the variance capability of the production or increases costs since either the entire assembly line needs to be larger or several different assembly lines are needed to accommodate the part variations. Part of the reason for creating a new material delivery system is that the company would like to include more variations in their products without increasing costs.

One main part of the delivery system is the Autonomous Transport Robots (ATR) that deliver assembly kits to the production line. There are several challenges with this new concept, including path following, and considering *Just in time* (JIT) process constraints. Solving these challenges will form the basis for this master thesis.

The goal is to develop a path following system that utilizes Model Predictive Control (MPC). The communication part of the system will be implemented using the Data Distribution Service (DDS) and Robot Operating System (ROS) 2.0 framework. The ATRs perception input will be provided from cameras on the factory ceiling as well as the robot's wheel sensors.

The development of this project aims to cover current knowledge gaps in the implementation of MPC for path following for ATRs. Such as investigating the technical feasibility of using MPC controllers for path following in a factory environment. Nowadays, ATRs are being used more often in production settings. This setting puts different demands on the ATR's performance compared with autonomous vehicles in other environments. Requirements on precise delivery punctuality and collision avoidance are important demands to consider. A robust path following system is fundamental to fulfill these requirements. Since humans are working in the same area as the ATRs, safety is of big concern. Therefore, the capability to limit the robot's speed is needed. This poses further constraints on the punctuality of the system, since limiting the robot's speed would then make it harder to reach its target location in the desired time frame.

1.2 Related work

In the master thesis *Obstacle Avoidance for Mobile AGVs Using a Photogrammetry-Based Sensor System* [5], A. Cavin used a photogrammetry-based sensor system (GPSS) together with odometry to localise and drive a single robot along with a trajectory. Cavin also evaluated different types of observers for fusing odometry and photogrammetry data.

Using a Kalman filter for localization and sensor fusion of odometry and photogrammetry is also investigated and proven to be an accurate method in the article *Fusing vision and odometry for accurate indoor robot localization.* [3]

The work of K. Pålsson and E. Svärling in the master thesis *Nonlinear Model Predictive Control for Constant Distance Between Autonomous Transport Robots* [11] is, in turn, an extension of [5]. Pålsson and Svärling tried to use a nonlinear MPC controller to keep simulations of two robots in constant distance while being guided by GPSS. Their thesis also compare the three different solvers, Fmincon, IPOPT, and PANOC. Pålsson and Svärlings' result implies that it is feasible to implement a similar path following system as an embedded controller.

The solving algorithm recommended by Pålsson and Svärling, PANOC, is introduced and presented in *A Simple and Efficient Algorithm for Nonlinear Model Predictive Control.* [14] The PANOC alsorithm is also evaluated and proven to be a good solution for an embedded path following control system in the article *Embedded nonlinear model predictive control for obstacle avoidance using PANOC.* [12]

1.3 Objectives

The goal of this thesis work is to design and implement an MPC controller for an ATR. The MPC controller makes sure that the ATR follow a given trajectory so that the delivery of materials meets a specific deadline. It is also important that it reaches each point in the path at the correct time as to avoid any collisions. The objective is to answer the following research questions:

- What is the technical feasibility of using embedded MPC controllers for ATRs as material delivering systems?
- How well does an MPC path following system for an ATR perform when it comes to meeting the system deadlines of a production line?

1.4 Scope

Much work has already been done in the development of material delivery systems. The scope of this master thesis work is to focus on realizing the actual implementation of the controller on the ATRs. This also means that some part of the process will be used from the previous work [5], and [11].

The path following system will mainly focus on the specific ATRs provided by Volvo. This means that the evaluation process will only focus on how the developed algorithm works on this specific platform and will disregard how it would function on any other systems with different specifications. For example, different speed restrictions, size, and actuator performances. Also, the reference path followed as well as the photogrammetry poses will be provided from other parts of the delivery system and is not included in the scope of this project.

1.5 System overview

The delivery system is divided into four modules: visual tracking, obstacle avoidance, fleet control and path following.

The visual tracking system uses the feed from cameras mounted on the ceiling to locate robots and obstacles. This information used by the obstacle avoidance system to generates a collision-free trajectory for each ATR. The fleet control system takes that trajectory and makes sure that there are no collision between any robots in the system. If a collision is detected, the trajectories of affected robots are altered in time, but not in space. The trajectory is then fed to the path-following system that will try to follow the path to the highest possible precision.

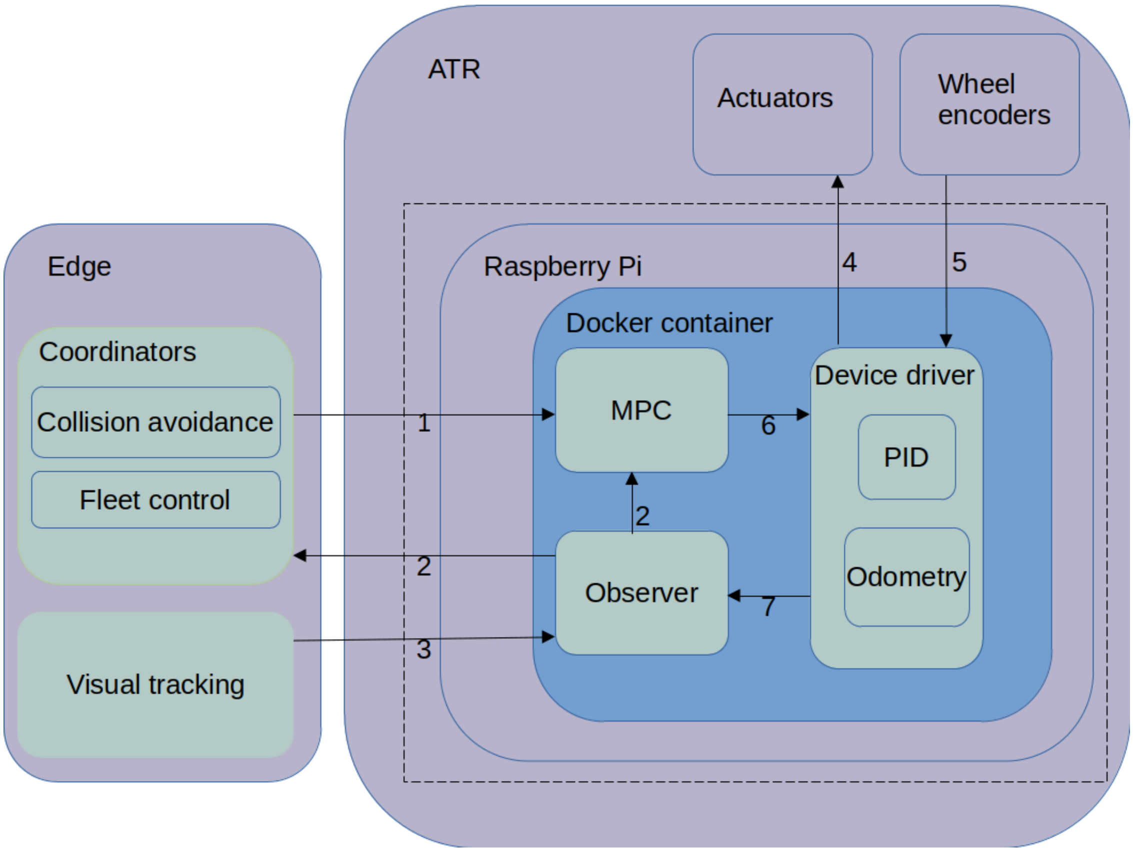

Figure 1.1: Diagram of the path following systems architecture and its communication with other modules

Each part of the delivery systems is depicted by Figure 1.1. The dashed line in the figure shows the boundaries of the path following system. It may be noted that all parts, except for the path-following system, are implemented on other computers on the network (edge). The path-following system is implemented on a single-boarded computer embedded on every ATR for faster response time. It should also be noted that the entire path following module is implemented in a docker container on a Raspberry Pi to enable faster deployment.

#	Signal
1	Reference trajectory
2	State estimation
3	Photogrammetry
4	Power command
5	Wheel ticks
6	Velocity command
7	Odometry

Table 1.1: Signals in Figure 1.1

The signals from Table 1.1 refer to their corresponding numbers in Figure 1.1. The first signal is the reference trajectory that is sent to the Model Predictive Controller (MPC). The MPC uses signal 1 together with signal 2, the state estimation, to find an optimal velocity for the ATR. The MPC sends this optimal velocity, as represented by signal 6 in Figure 1.1, to the device driver. In the device driver, the PID controller uses the velocity command to compute the error between the new and current velocity. The PID also uses old velocity errors as integral or derivative errors. These errors are then multiplied by their respective coefficients, which depend on the PID tuning, and summed up. This sum results in a power command that is transferred as signal 4 to the actuators. The device driver also receives signal 5, containing wheel ticks, from the wheel encoders. These wheel ticks are transformed into the delta positions Δx, Δy, and $\Delta\theta$, which are sent as odometry data to the observer via signal 7. The observer also takes photogrammetry data as input from the visual tracking software in the form position in x, y, and θ, as described by signal 3. The observer uses these inputs to create a state estimation that is then broadcasted as signal 2 both back to the MPC as well as to the coordinators.

This structure was chosen to increase modularity. The blocks in the docker container could be switched as long as the inputs and outputs stay the same. This means that the device driver can be exchanged to make the path following system usable with a different vehicle than the one it's currently designed for.

2
Localisation

The localisation of the ATR is devided into three different parts, the odometry, the photogrammetry, and the fusion of data received from these. Odometry works very well in a short term perspective while photogrammetry is not as precise, but does keep constant over long periods of time. By using both and estimating the actual state with a Kalman filter, a more robust localisation can be achieved.

2.1 Odometry

Odometry is the use of motion sensor data. This data can differ much in quality depending on the quality of the sensors used. In the short time, the odometry provides good data essential for the localization. However, as soon as the odometry runs for a more extended period, the problem of drift occurs [4]. The sources of this drift are, in part, limited sensor resolution, unequal wheel diameters, and wheel slip. Having drift in the system means that the odometry data is reliable only for a specific time before the error in the data becomes too large.

The odometry sensors on the ATR measures wheel ticks. The ticks are then multiplied with a distance per tick factor. This information is used to calculate how far each wheel has traveled since the previous measurement. With this information, dead reckoning can be used to estimate the pose of the vehicle. Dead reckoning is a method of estimating a position using a previous, fix, orientation and speed estimations. Dead reckoning is subject to cumulative errors, which need to be counteracted to avoid further drift in the system. The distance moved by each wheel is estimated as shown in Equation (2.1).

$$\begin{bmatrix} \Delta d_{left} \\ \Delta d_{right} \end{bmatrix} = \Delta n_{tick} W_{mpt} \tag{2.1}$$

Δn_{tick} is a vector with the number of ticks for each wheel since the last measurement, and W_{mpt} is a model-specific parameter describing the distance represented by each wheel tick. This information is then used to estimate the total distance traveled for the robot since the last estimation.

$$\Delta d = \frac{\Delta d_{left} + \Delta d_{right}}{2} \tag{2.2}$$

$$\Delta \theta = \frac{\Delta d_{left} - \Delta d_{right}}{W_{bw}} \tag{2.3}$$

Where W_{bw} is the robots wheel base width and Equation (2.4) describes the calculation of the vehicle orientation.

$$\theta = \sum_{i=1}^{N} \Delta\theta_i \tag{2.4}$$

$$\Delta x = \Delta d + cos(\theta) \tag{2.5}$$

$$\Delta y = \Delta d + sin(\theta) \tag{2.6}$$

The odometry data is then sent to the observer to be fused with the photogrammetry, to estimate the robots state.

2.2 Photogrammetry

The second localization method used to determine the pose of the robot was photogrammetry. Photogrammetry is a method used to get measurements from pictures. The pixels of the pictures are translated into euclidean distances. A fixed grid of cameras was mounted to the factory ceiling for this purpose. In total, these cameras will together provide near-complete coverage of the floor area. The ATRs have visual markers on them, making them identifiable. When a visual marker is detected, the pose of the ATR is calculated and sent to the robot's observer. The photogrammetry calculations was not included in the embedded path following system module, but provided from the vision module on the edge of the delivery system. The data provided by the photogrammetry is fused with the odometry for a combined estimate of the robot's pose.

When using photogrammetry, errors can occur due to moving objects being blurry, noise, lens distortion, and model parameter uncertainty [5]. The pose that is provided by the photogrammetry may not be as accurate as the odometry data. The photogrammetry poses will not be affected by any cumulative error, which means it will provide pose estimations without drifting. However, the poses will be subject to a more considerable delay than odometry since the position data will be extracted from a video stream, which takes longer than reading the odometry data.

2.3 Observer

With multiple sensor inputs on varying frequencies, an observer is needed to estimate the pose of the ATR. The observer takes the data provided by the odometry and photogrammetry and fuse these into a state estimation. This estimation is then passed on to the MPC. The odometry data is fast and updated with the same frequency as the regulators, in this case 100Hz. The photogrammetry data is updated at a slower rate and might be less precise, but it can be used to remove the cumulative error in pose caused by the odometry drift.

As concluded by [11] and [5], a linear Kalman filter was sufficient for this type of implementation, where the filter uses odometry data in the prediction step and photogrammetry data in the update step. The prediction step utilizes a model with the odometry as input. Because the sensor inputs from the odometry are faster than photogrammetry, there will often be multiple prediction steps for every update step. There might also arise situations where no photogrammetry data is available for a period of time.

The prediction model of the Kalman filter was modeled according to the linear state space model in Equation (2.7), as proposed by [5] and [3]. The systems state space matrices are $A = B = C = I_3$, and the system input u_k is the change in position and angle, $\begin{bmatrix}\Delta x & \Delta y & \Delta\theta\end{bmatrix}^T$, provided by the odometry from the wheel sensor data. $\hat{x}$ is the predicted states of the robot.

$$\begin{aligned} \hat{x}_{k+1} &= A\hat{x}_k + Bu_k \\ \hat{y}_k &= C\hat{x}_k \end{aligned} \tag{2.7}$$

The covariance matrix P is updated in every prediction step as described in Equation (2.8), where R is a covariance matrix that represents the uncertainty in the odometry. If a prediction is made multiple times before the next update step, the covariance is increased for every prediction.

$$P_k = AP_{k-1}A^T + R \tag{2.8}$$

The update step is made every time the observer gets input from the photogrammetry. Firstly, by calculating the Kalman gain, K , as in Equation (2.9), where Q is a covariance matrix, similar to R, that represents uncertainty in the photogrammetry.

$$K_k = P_kC^T[CP_kC^T + Q]^{-1} \tag{2.9}$$

Secondly, the predicted state is updated. The state estimate is corrected by the photogrammetry input, $\bar{x}$. The error between the state estimate and the camera input, scaled by the Kalman gain, updates the prediction according to Equation (2.10).

$$\hat{x}_{k+1} = \hat{x}_k + K_k(\bar{x} - C\hat{x}_k) \tag{2.10}$$

Finally, the covariance is updated as described in Equation (2.11)

$$P_{k+1} = (I - KC)P_k \tag{2.11}$$

All matrices describing the prediction model are identity matrices, hence the algorithms for the prediction and update can be simplified as shown in Algorithm 1.

Algorithm 1: Kalman filter

if *Prediction* **then**
- $\hat{x} \leftarrow \hat{x} + u$
- $P \leftarrow P + R$

else if *Update* **then**
- $K \leftarrow P(P + Q)^{-1}$
- $\hat{x} \leftarrow K(\bar{x} - \hat{x})$
- $P \leftarrow (I - K)P$

The uncertainty matrices Q and R can be tuned to reach a specific behavior in the observed states, depending on how much we want to trust the data from the odometry and the photogrammetry.

In the case where the latency in the photogrammetry information is large enough to cause problems, the localization system will be unreliable and cause problems for the control system. To counteract this delay, the look-and-move approach was implemented. This approach means that the robot has to stand still for a specified period of time before using the photogrammetry data [10]. It is important that the time standing still before updating is longer that the time delay in the measurement, to make sure that the photogrammetry position catches up with the true position of the robot.

The uncertainty matrices Q and R may need a different tuning when using the look-and-move approach, compared to using continuously updates for the photogrammetry. During the evaluation of the path-following system both Q and R where chosen to be identity matrices.

3
Path following control system

The path following controller was divided into two separate controllers. The first step is an MPC regulator that relies on a motion model based on the robot's angular and linear velocity. The second step is a PID regulator that yields a power command to the actuators and makes sure that the target velocities from the MPC regulator are followed, as illustrated by Figure 3.1.

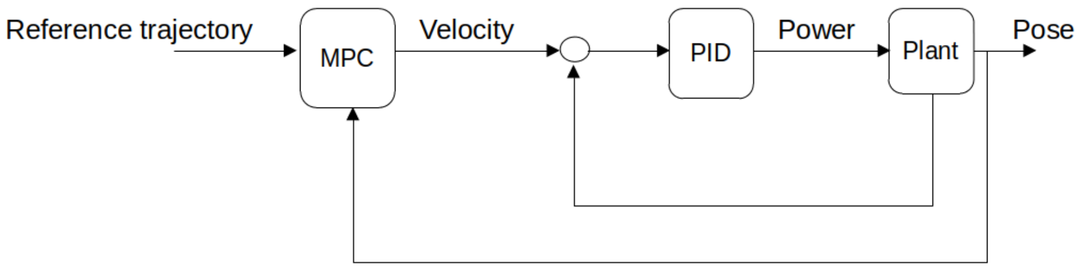

Figure 3.1: Block diagram over the two-step control system

3.1 Nonlinear Model Predictive Control

Model Predictive Control (MPC) is an optimization-based control technique. One of the main advantages of using MPC is the possibility of defining constraints on the system behavior. The constraints can be specified on both the inputs and outputs of the process. MPC is also called receding horizon control, and it can be compared to the Linear Quadratic Regulator (LQR). The LQR approach is to find the optimal control for a quadratic optimization problem with an infinite time horizon. The horizon is the main difference from MPC, where the optimization problem has a finite time horizon N. Another essential difference is that the optimization problem in the MPC is solved online, in every time instance. This requires a fast enough solve time of the target optimizing problem. This solving time can be a limitation in the control system's performance.

A few of the important ingredients needed for a MPC controller [6] are:

- An internal model that describes the physical process
- An objective to express desired system behaviours
- The receding horizon principle since the first control action is applied to the process

These ingredients are used as an iterative process to define and solve the optimization problem. The MPC optimizes the control input for the entire horizon and then applies the next input to the plant. The process is then done again for every time step using the receding horizon approach.

The most computationally efficient type of MPC is linear [7]. For an MPC to be linear, the system model, cost function, and constraint all have to be linear as well. When one or several of these parts cannot be linearized, a nonlinear version of MPC (NMPC) can be used instead. An NMPC problem is harder to solve and takes a longer time. As shown by [11], there have been breakthroughs where new solvers can significantly reduce this time enough to be used for the path following control problem.

3.1.1 Motion model

A model of the system is an important part of the MPC scheme. One of the advantages of nonlinear MPC is that a nonlinear model can be utilized. The model used in the MPC is based on the Equation (3.1). x and y represent the position of the robot while θ is the orientation. This is illustrated in Figure 3.2.

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} cos(\theta)v_\theta \\ sin(\theta)v_\theta \\ \omega \end{bmatrix} \tag{3.1}$$

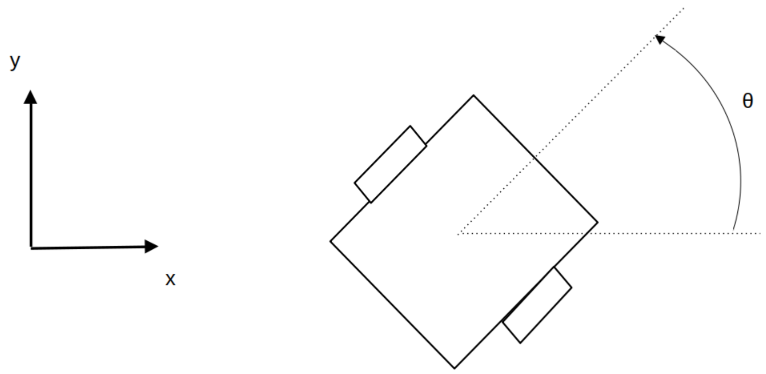

Figure 3.2: The robot states in a global coordinate system

The system model's continuous dynamics were discretized using the Euler method as described in Equation (3.2), where h is the discrete step length parameter. This yields the discrete motion model in Equation (3.3) that was used in the prediction phase of the MPC.

$$x_{k+1} = x_k + hf(x_k) \tag{3.2}$$

$$\begin{bmatrix} x_{k+1} \\ y_{k+1} \\ \theta_{k+1} \end{bmatrix} = \begin{bmatrix} x_k + h\ cos(\theta_k)\, v_k \\ y_k + h\ sin(\theta_k)\, v_k \\ \theta_k + h\ \omega_k \end{bmatrix} \tag{3.3}$$

This model was based on a robot with differential steering, but changing the model to describe the dynamics of another vehicle is possible. This makes the system generic and modular, but it may be needed to tune the parameters differently in the MPC formulation to achieve a similar behavior.

3.1.2 MPC formulation

Figure 3.3 illustrates how the path following control problem was modeled. The reference signal for the regulator to track is a sequence of poses in the 2D space. Each pose is also connected to a particular time, as defined in Equation (3.4). N is the number of discrete time steps that define the horizon length of the optimization problem, together with the step length parameter h.

$$t_{ref}, [x, y, \theta]_{ref} = \begin{bmatrix} t_k, [x, y, \theta]_k \\ t_{k+1}, [x, y, \theta]_{k+1} \\ \vdots \\ t_{k+N}, [x, y, \theta]_{k+N} \end{bmatrix}_{ref} \tag{3.4}$$

The control signal that the regulator generates consists of the robot's angular velocity, $\dot{\theta}$, and the linear velocity relative to the orientation, v_θ. The current pose of the robot is measured and sent back to the regulator in a closed-loop system.

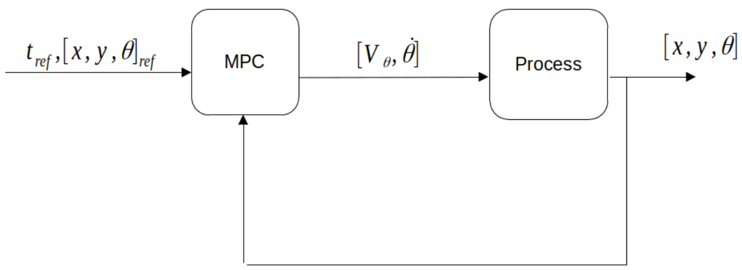

Figure 3.3: Block diagram of the MPC system

The formulation of the optimization problem was defined as in Equation (3.5). $x - r$ is the state vector error $[x - x_{ref}, y - y_{ref}, \theta - \theta_{ref}]$. u is the speed and angular velocity $[v_\theta, \omega]$, and $\dot{u}$ is their respective derivatives $[\frac{v_k - v_{k-1}}{h} \frac{\omega_k - \omega_{k-1}}{h}]$. Constraints were defined on the speed and angular velocity to make sure that the control signals stay within the robots limits during the horizon length N.

$$\begin{aligned} \min_u \quad & \sum_{k=0}^{N} (x - r)_k Q (x - r)_k + u_k R_1 u_k + \dot{u}_k R_2 \dot{u}_k \\ \text{subject to} \quad & u < u_{max} \\ & u > u_{min} \end{aligned} \tag{3.5}$$

The weights Q, R_1, and R_2 are diagonal matrices that determines the importance of minimizing the different variables in the problem. They can be seen as user defined tuning parameters and was tuned to give the path following system the desired behavior. The chosen tuning parameters are displayed in Table 3.1, together with the control signal constraints u_{min}, u_{max}, the step length h and the horizon length N.

Q	$\begin{bmatrix} 150 & 0 & 0 \\ 0 & 150 & 0 \\ 0 & 0 & 25 \end{bmatrix}$
R_1	$\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$
R_2	$\begin{bmatrix} 10 & 0 \\ 0 & 1 \end{bmatrix}$
N	20
h	0.01
u_{min}	$\begin{bmatrix} 0 \\ 0 \end{bmatrix}$
u_{max}	$\begin{bmatrix} 0.6 \\ 1 \end{bmatrix}$

Table 3.1: MPC parameters

3.2 PID

The second part of the control system is a proportional–integral–derivative (PID) controller. Two advantages of a PID controller are its simplicity and the fact that no plant model is needed [16]. The PID controller receives its reference signal from the MPC in the form of a velocity command vector. This velocity vector includes a velocity in the θ direction, v_θ, and an angular velocity around the robot's $z - axis$, $\dot{\theta}$. These velocities were transformed into the left and right wheel speed according to Equation 3.6.

$$\begin{bmatrix} v_{LW} \\ v_{RW} \end{bmatrix}_{ref} = T \begin{bmatrix} v_\theta \\ \dot{\theta} \end{bmatrix}_{ref} \tag{3.6}$$

where $T = \begin{bmatrix} 1 & -\frac{W}{2} \\ 1 & \frac{W}{2} \end{bmatrix}$ and W is the wheel base width of the robot.

The wheel speed from Equation (3.6) is then used by the PID controller where the error e is defined as the difference between the reference wheel speed and the wheel speed measured by the odometry, as is shown in Equation 3.7.

$$e(t) = \begin{bmatrix} v_{LW} \\ v_{RW} \end{bmatrix}_{ref} - \begin{bmatrix} v_{LW} \\ v_{RW} \end{bmatrix}_{odom} \tag{3.7}$$

This error is then used to define the integral and derivative errors

$$e_i(t) = \int_0^{t_k} e(\tau) d\tau \tag{3.8}$$

$$e_d(t) = \frac{de(t)}{dt} \tag{3.9}$$

Discretizing the errors yields the Equations (3.10), (3.11) and (3.12).

$$e_p(t_k) = e(t_k) \tag{3.10}$$

$$e_i(t_k) = \sum_{i=1}^{k} e_p(t_i)\Delta t \tag{3.11}$$

$$e_d(t_k) = \frac{e_p(t_k) - e_p(t_{k-1})}{\Delta t} \tag{3.12}$$

By combining these discretizations of the errors, the algorithm describing the PID regulator can be written as the Equation (3.13).

$$u = K_p e_p + K_i e_i + K_d e_d \tag{3.13}$$

K_p, K_i, and K_d are tuning parameters for the controller and u is the power commands, $\begin{bmatrix} P_{LW} & P_{RW} \end{bmatrix}$, for the left and right wheel, that is sent to the plant. The tuning was made manually to achieve a desired behavior and the parameters was chosen as $K_p = 200, K_i = 0, K_p = 4$. Removing the integral part means that the controller actually is a PD controller. Figure 3.4 is an overview of the closed loop control system described by this chapter.

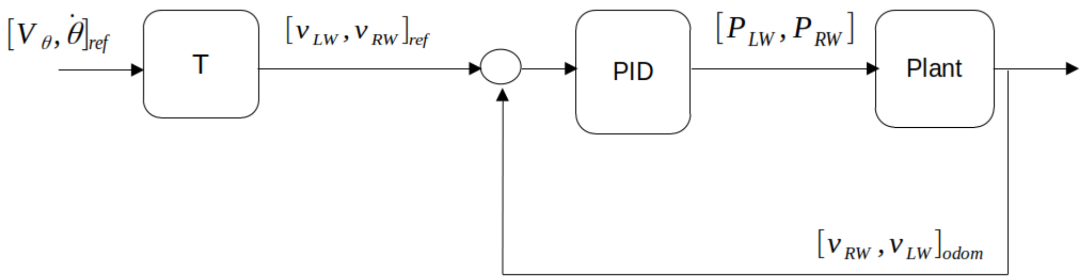

Figure 3.4: Block diagram of the PID regulator system

3.3 System architecture

The complete system is composed of the integration of the modules MPC, PID and observer, shown in Figure 3.5. It can be seen as a two-step control scheme, where the reference signal into the system is the pose for every time step in the receding horizon. The MPC is the outer loop and delivers the reference signal for the PID controller in the inner loop. The PID controller feeds the plant the actual power command for the actuators.

The observer takes input from the odometry sensors on the robot but also from the external camera system. The estimated state from the observer is used by the MPC, while the PID controller only uses sensor readings directly from the odometry. The system's inner and outer loops are designed to run with a higher updating frequency than the inputs from the external camera system and reference trajectory modules, which can be less frequent.

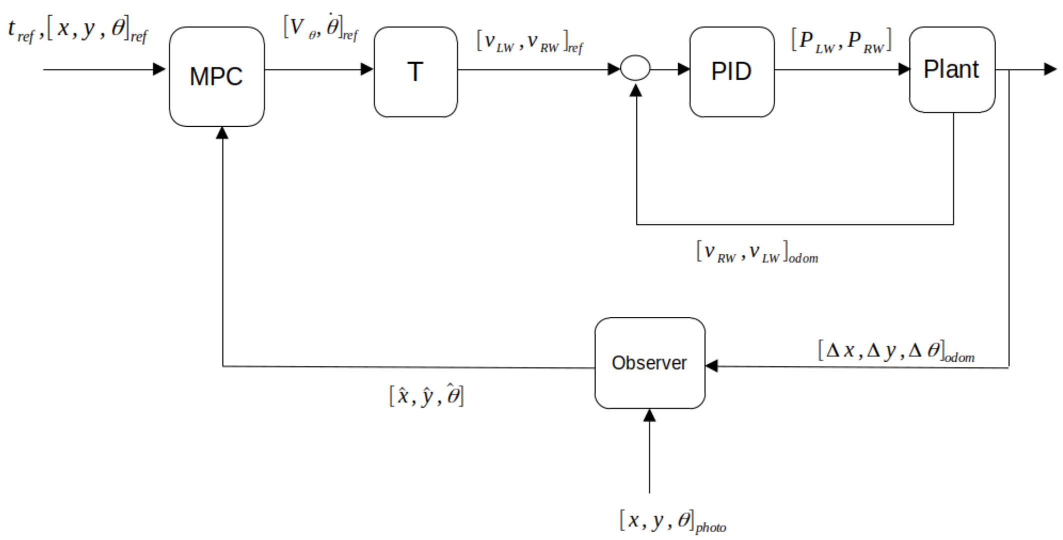

Figure 3.5: Block diagram of the full path following control system

4

Deployment and communication

The control system was deployed onto the hardware using docker. To communicate internally on the embedded system and with the edge, the robot middleware ROS2 was utilized. Optimization Engine was used to generate a fast solver for the embedded MPC.

4.1 Hardware

For evaluating the path following system, an ATR is used based on a slightly modified Husqvarna 450X Automower. The power has been rerouted to accommodate a single-board computer so the computer can remain on even when the Automower is charging. The single-boarded computer used to control the Automower is the Raspberry Pi with 8GB of SDRAM and a quad-core processor with the clock speed 1.4GHz. During some of the tests, the Raspberry Pi had a heat sink and fan connected. The ATR is depicted in Figure 4.1. The Automower battery powers the Raspberry Pi via a 5-volt connector. They are also connected from a USB A port on the Raspberry Pi to a USB B port on the Automower motherboard.

Figure 4.1: The autonomous transport robot

As can be shown in 4.1, the ATR also has a mounted table with a visual marker on it. The visual marker helps the visual tracking part of the delivery system determine the pose of the ATR, and the purpose of the table is to carry the load.

4.2 Docker

Docker is a service that virtualizes an operating system. It uses this virtualization to deliver software in easily accessible packages [9]. A package or container, as Docker calls it, can be built on any computer and still work when deployed onto the Raspberry Pi, which has another computer architecture. The container can then be started and perform just like every other container active on any other platform that it was cross-compiled to function on.

The Docker service is helpful since it can be used for fast upscaling by deploying the software to each Raspberry Pi with the click of a button. Every dependency is pre-built into the container. It also removes the need to build and compile any other software on the Raspberry Pi by deploying these containers. Using Docker may save building time on a platform with little computational power, such as the Raspberry Pi.

4.3 ROS2

The Robot Operating System 2 (ROS2) was used for communication between the MPC, the PID controller, and the observer. ROS2 was also used for communication with the external parts of the delivery system, both for visual tracking input and the reference trajectory. The ROS2 middleware is an assortment of tools and libraries made for software development of robotics systems [8]. Using ROS2 also adds modularity, where a module with a ROS2 interface easily can be exchanged as long as the input and output stay the same.

The communication in the delivery system consists of subscribers, publishers, clients, and servers. Each of these communication types uses a pre-defined message type to ensure that both the sender and receiver know what is included in the message. All the communication types use a topic name that guides the message to the correct receiver. The subscribers subscribe to a specific topic name, and whenever a message is sent on that topic, they will read the data from it. In the same spirit, the publisher will send a message on a topic for any subscribers listening. The clients and servers work differently, where first, a request message is sent from the client to the server. The server then responds depending on the information in the request. Therefore, the server-client communication type offers a guarantee that the message communication is synchronous. In contrast, the subscriber-publisher does its communication asynchronous. Figure 4.2 illustrates the ROS2 communication and the modularity of the system.

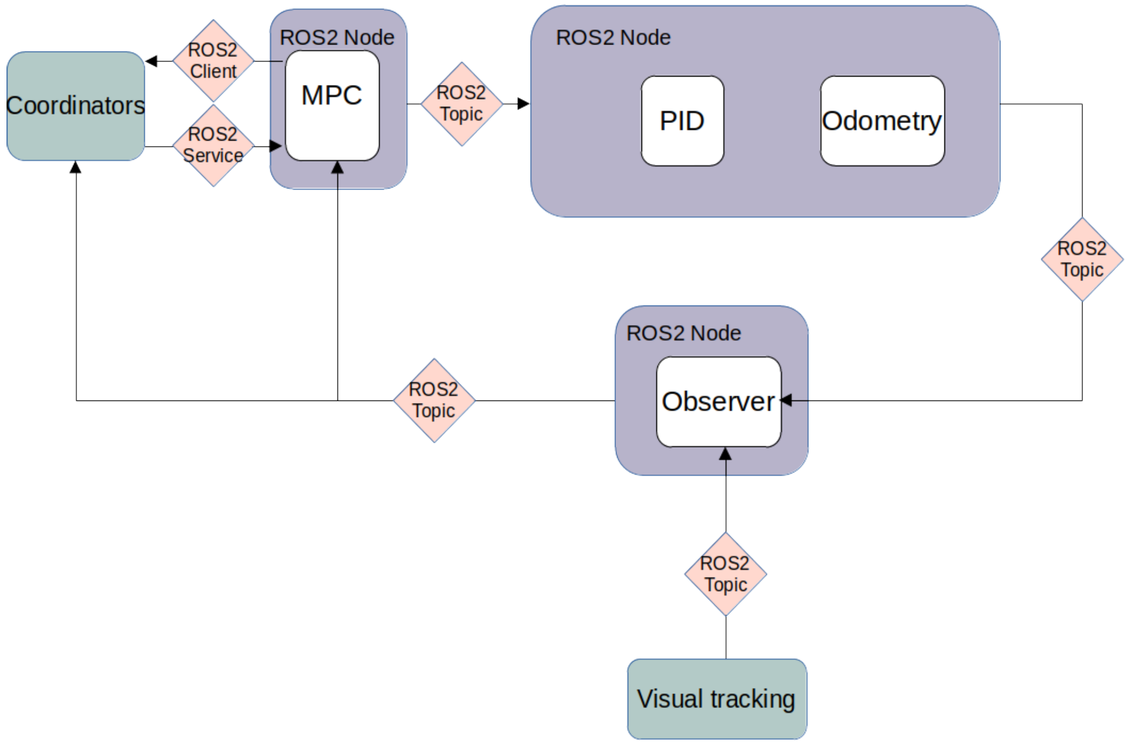

Figure 4.2: Illustration of the ROS2 communication

4.4 Optimization Engine and PANOC

PANOC is a method for Optimal Control used to solve non-convex optimization problems using Newton-method and forward-backward iterations. These methods make the PANOC a fast solver, able to achieve sub-millisecond-fast solve times for MPC [15] and close to 200 times faster solve times for NMPC than other methods such as IPOPT or Fmincon [11]. Keeping a rapid update rate for the MPC is crucial in a path-following system, where reactions to potential changes in the environment are essential.

To include the PANOC solver into the NMPC, Optimization engine (OpEn) [13] was used. OpEn is software that uses the symbolic framework CasADi [1] to define the optimization problem and its constraints. The optimization engine then generates an optimizer in rust code suitable for embedded control. This rust code was then deployed together with all other code using Docker.

5

Evaluation

The evaluation of the path-following system was divided into three categories. The control and localization was evaluated based on different setups and trajectory scenarios, and the computational efficiency of the MPC solver and embedded computer was assessed. Also, the path-following systems performance together with the fleet control system was tested, to ensure collision avoidance between robots.

5.1 Localization and control

To evaluate the accuracy in measurements and the control system performance, three different setups were used. Firstly, the robot was pushed by hand along a pre-defined trajectory on the ground. In the second setup, the path-following control system was used, with only odometry data as input. For the third test, the look-and-move approach was utilized.

The same two reference trajectories were used for all three test cases. The trajectories are depicted in Figures 5.1 and 5.2. The first one is a rectangle with a width of 5 meters and a height of 2. The second path is based on a rectangle with the same dimensions, but this time the corners were taken very smooth, giving the trajectory an ellipse-like shape. Both cases followed the trajectory in an anti-clockwise fashion.

The two cases with different corners were defined to see how the localization measurements and the controller would be affected by different types of turns. The rectangle simulates a harder type of scenario since the smooth corners are easier for the robot to handle.

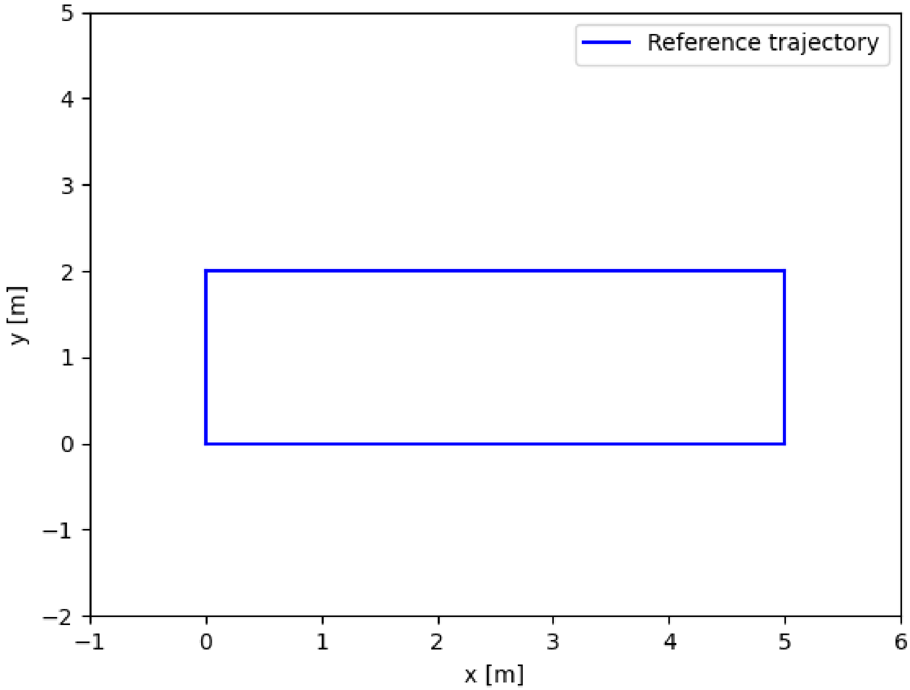

Figure 5.1: Rectangle reference trajectory

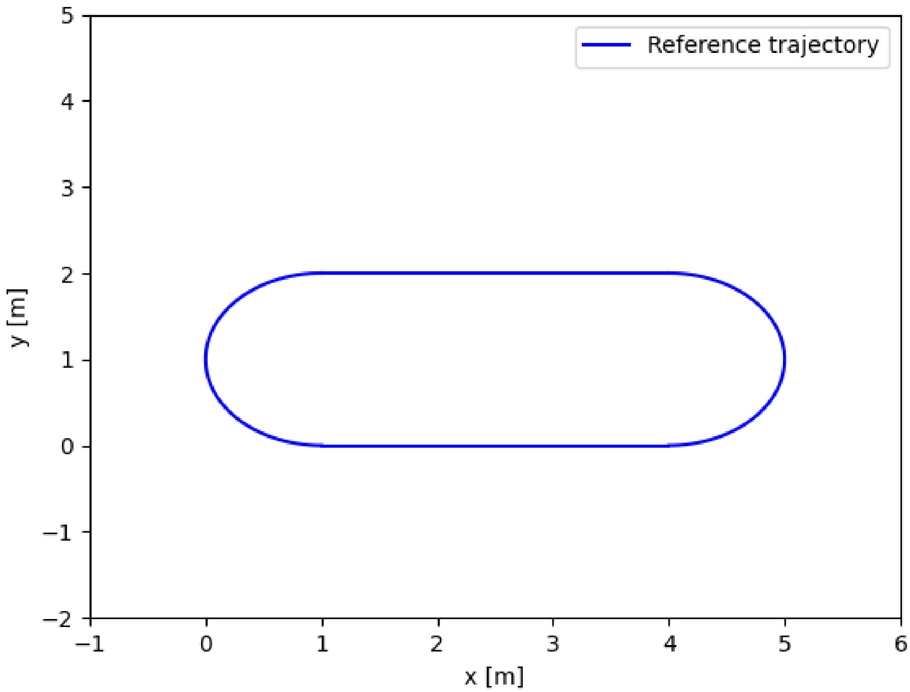

Figure 5.2: Ellipse-like reference trajectory

For the first setup, where the robot was pushed by hand, only the odometry data was measured. The pre-defined trajectories were marked on the ground so the ATRs position, estimated by with odometry, could be compared to ground truth. This test is supposed to give information about how large the accumulated error in the odometry becomes over time, which determines how often the photogrammetry input is needed and how much the observer can trust each sensor input. It is also important to for other modules of the delivery system to know how much the position data can be trusted, to create reference trajectories with enough safety marginal. The trajectories was followed for ten laps, with a speed of approximately $0.5 m/s$.

The second test was run with the path following controller. The sensor data from the odometry and the photogrammetry position were both measured but the controller used only on input from the odometry. This data can then be used to assess the delay in the photogrammetry system and its precision. It is important to know the precision to determine how much the observer can trust the data. But even more important is to evaluate the latency of the photogrammetry. A low latency is a basic condition for fusing the photogrammetry and odometry continuously, without using the look-and-move approach. This test case was also performed to evaluate how well the control system can track the reference trajectories. The trajectories was followed for five laps, with a maximum speed of approximately $0.25 m/s$.

The control system was also used to run the ATR in the third setup. But this time using both odometry and photogrammetry measurements. To counteract the delay in photogrammetry, the look-and-move method was used. A short stop was introduced in the trajectories at the beginning of each lap, and the observer was only allowed to use the photogrammetry after standing still for more than three seconds. The delay of the photogrammetry measurements was approximated be around two seconds. The rest of the time was only the odometry used. The purpose for this test was to evaluate the performance of the full path-following control system and how well the odometry drift can be counteracted. Also here, the trajectories was followed for five laps, with a maximum speed of approximately $0.25 m/s$.

5.1.1 Computational efficiency

A set of test scenarios was performed to assess how the embedded computers' computational efficiency affects the solve time of the MPC and the frequency of the control system. The robots wheels was lifted of the ground during all the tests and the robot was not actually moving. The first test case was run only with the essential processes on the Raspberry Pi. After the first test, it was rerun with two other ROS2 MPC nodes opened to minimize the computational power left for the steering controller, trying to simulate a worst-case scenario. The MPCs was given a reference trajectory with many discontinuities and variations to make the optimization problem harder to solve. The path can be seen in Figure 5.3.

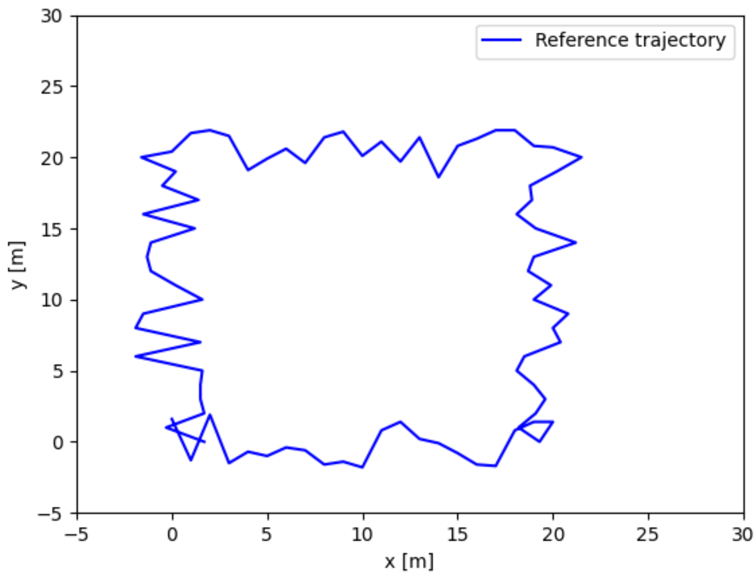

Figure 5.3: Reference trajectory for computational efficiency tests

The test was run during a long period of time to more accurately be able to gauge the temperature's effect on the solve time for the MPC. The evaluation was also done both with the cooling fan and heat sink attached as well as without, so a comparison can be made. The Raspberry Pi has a feature called thermal throttling, where it underclocks its processor to lower its temperature when it heats up to above 80 degrees Celsius, where a fan may increase the throughput by up to 90% [2]. This means that the cooling system may directly impact the computational efficiency and, in turn, might increase the possibility of using an embedded computer for this path following system.

5.1.2 Collision avoidance

An important reason for the precision demands on the path-following system is that the fleet control and obstacle avoidance systems need to decide each ATRs position in time accurately. One test was performed to evaluate the fleet control together with the path-following system, where the two reference trajectories depicted by Figure 5.4 was used.

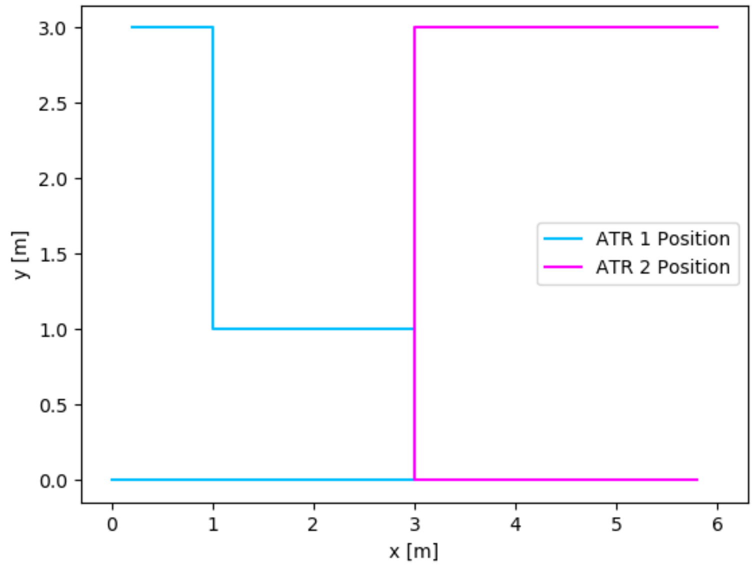

Figure 5.4: The reference trajectory before fleet control changes

The fleet control system then changed this trajectories to avoid any distance between ATRs lower than 2 meters. The distance in time before being changed by the fleet control system can be seen in Figure 5.5

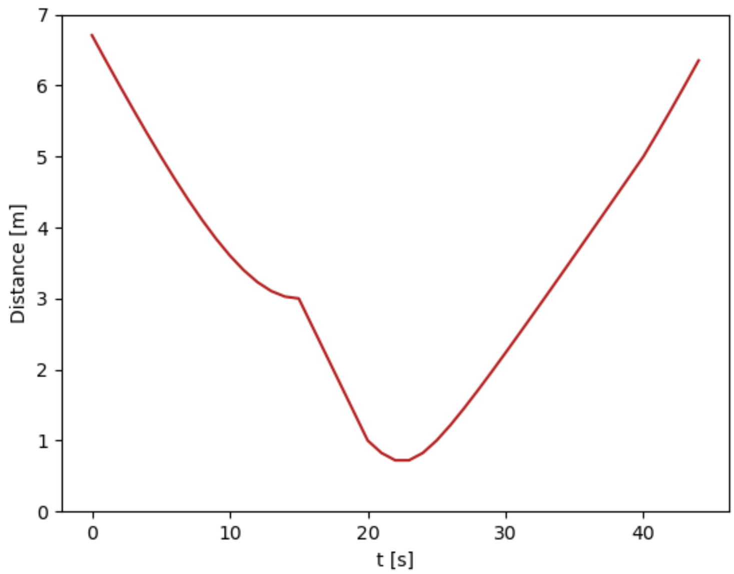

Figure 5.5: Distance between two ATRs according to fleet control reference trajectory

After the generation of the trajectory, the fleet control sends reference commands to the path-following system. It then starts to find possible dangerous intersections between paths and changes the speed of the ATRs to try to avoid collision.

6

Results

6.1 Odometry drift

Both test scenarios where the robot was pushed by hand along the reference trajectory showed a significant drift in position. During the test where the ATR was pushed in the rectangle trajectory, the estimated position drifted away for every lap. After 10 laps, with the ATR back at the origin, the odometry displayed the position (0.261, -0.462), which is a difference of 0.531-meter from the ground truth after 140 meters of travel.

In Figure 6.1, the odometry data is shown from where the ATR was pushed. The blue line depicts the first lap around the track, the red line shows the last lap, and the gray transparent lines illustrate everything in between. While the red line ended 0.531 meters from the ground truth as mentioned above, the first lap ends in $(-0.027, 0.035)$, which is approximately an 0.044-meter drift.

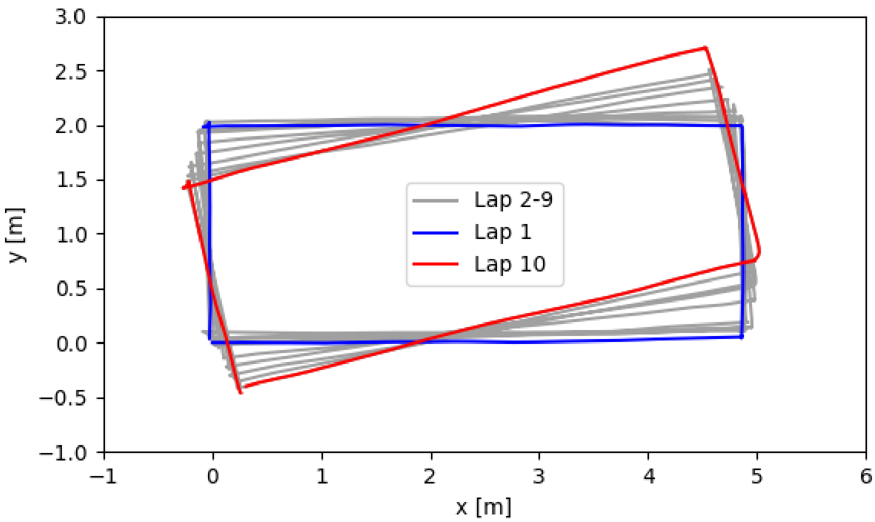

Figure 6.1: Odometry positions from the rectangle trajectory

Figure 6.2 shows the corresponding results for the ellipse-like trajectory. Also, here we can see a cumulative drift in the odometry. The error after the first lap was amassed to 0.390 meters since the ATR ended in the position (0.611, -0.027) compared to the actual position (1, 0). After the entire ten laps, the error was instead 1.235 meters since the positioning of the robot was now (2.512, 0.195)

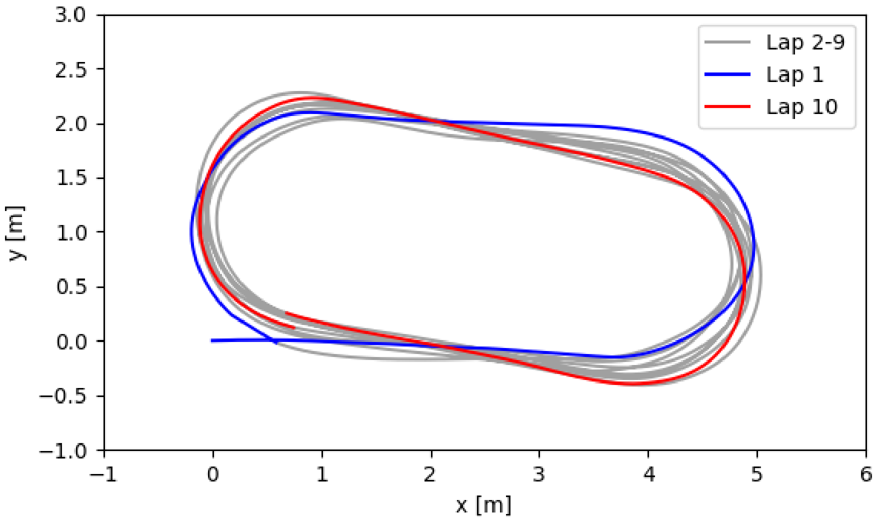

Figure 6.2: Odometry positions from the ellipse-like trajectory

6.2 Photogrammetry and odometry compared

Figure 6.3 shows the estimated position from the observer with only odometry as input, compared to the measured position from photogrammetry. It can be noted that the difference is growing more significant as the two measurements are drifting apart for every lap. The coordinate system is based on the observer.

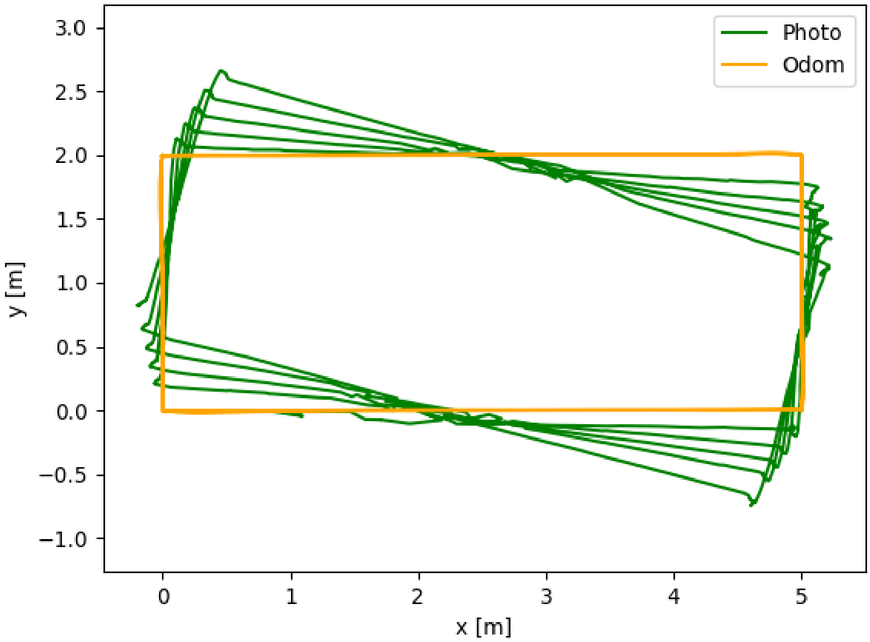

Figure 6.3: Position of the ATR based on photogrammetry and odometry in the rectangle scenario.

When comparing the two measurements x- and y-positions in time, the photogrammetry data are approximately two seconds behind the odometry. This is shown in Figure 6.4.

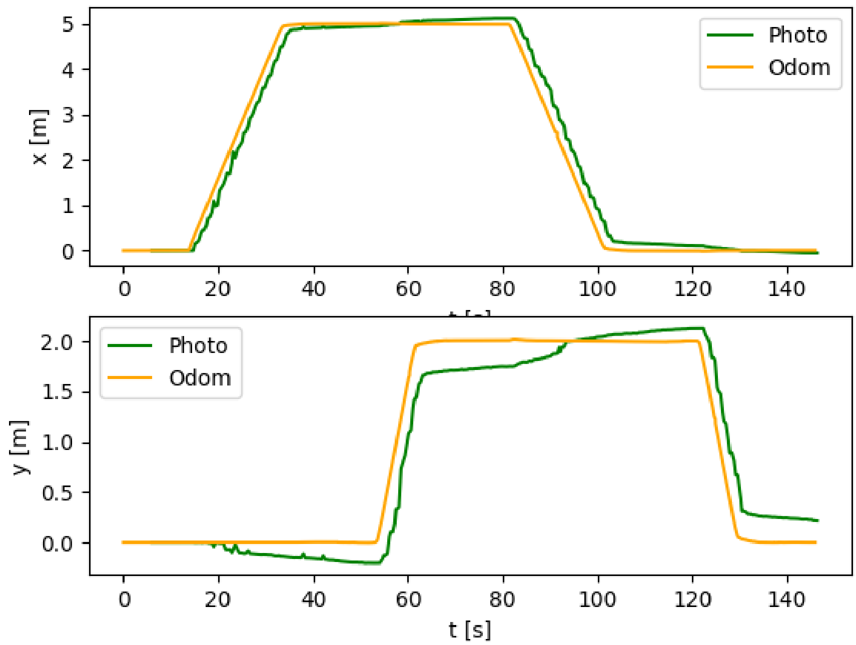

Figure 6.4: x and y positions over time for the photogrammetry and odometry in the rectangle scenario.

Figure 6.5 and 6.6 shows the corresponding data for the case with the ellipse-like reference trajectory. The same behavior as for the rectangle scenario can be noticed here. The trajectories based on the two measurements are drifting apart. The delay between the measured photogrammetry position and the estimated odometry is approximately two seconds.

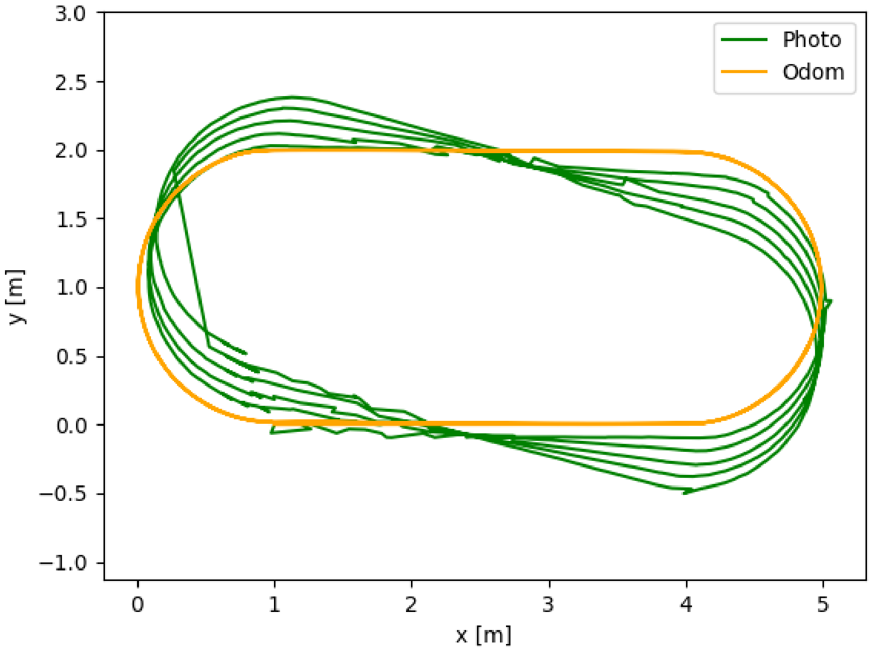

Figure 6.5: Position of the ATR based on photogrammetry and odometry in the ellipse-like scenario.

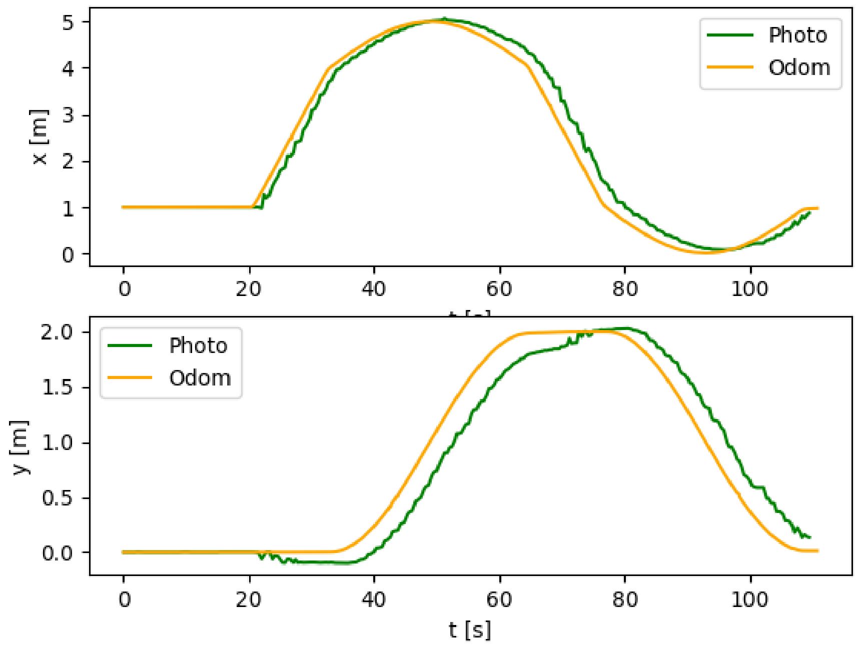

Figure 6.6: x and y positions over time for the photogrammetry and odometry in the ellipse-like scenario

6.3 Look-and-move

The results from testing the look-and-move method shows that the control system was able to remove any cumulative errors from the odometry by using the photogrammetry only after standing still for three seconds or more. As depicted by Figure 6.7, there is an error in the positioning of the observer compared to the photogrammetry measurements. However, the error does not grow larger over time. It can also be noted that there is an discontinuity in the observer position at the start of each lap. This is where the robot stands still and the position is corrected with photogrammetry data.

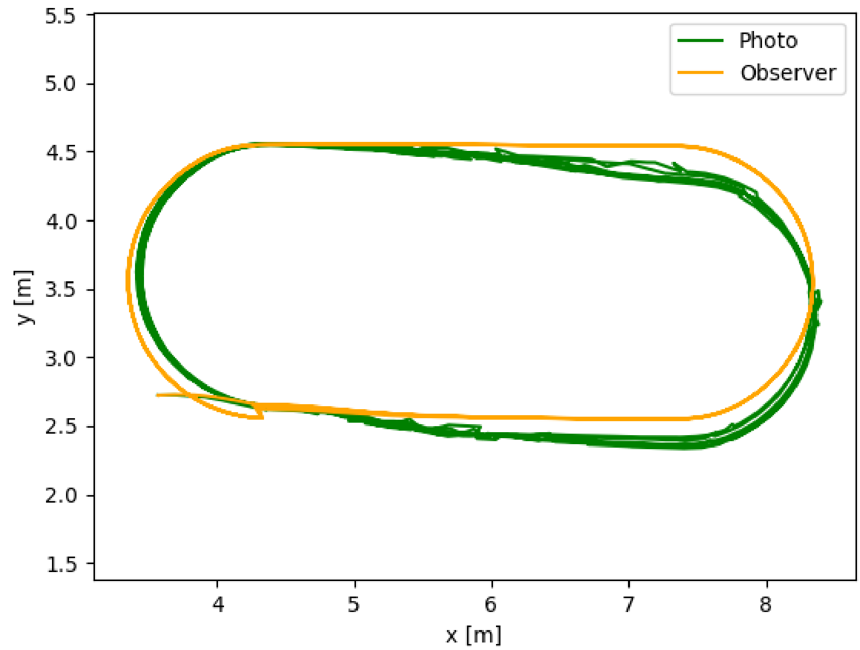

Figure 6.7: Position of the ATR when the controller is following the ellipse-like trajectory and using the look-and-move method.

Figure 6.8 shows the x and y position of the robot over time. The data shows that the photogrammetry has a delay compared to the odometry. However, when the robot stops at the end of the lap, the photogrammetry catches up to the observer position and, after standing still for a few seconds, the accumulated error is removed.

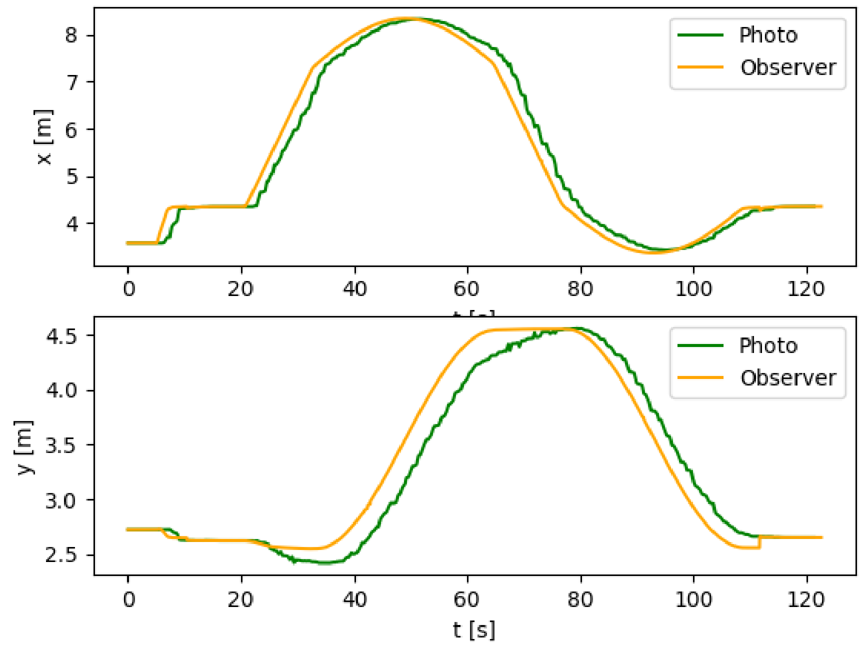

Figure 6.8: x and y positions of the ATR over time in the ellipse-like trajectory. The observer is using the look-and-move method.

Figure 6.9 shows the corresponding results from the rectangle reference trajectory. Also here the absence of a cumulative error can be noted.

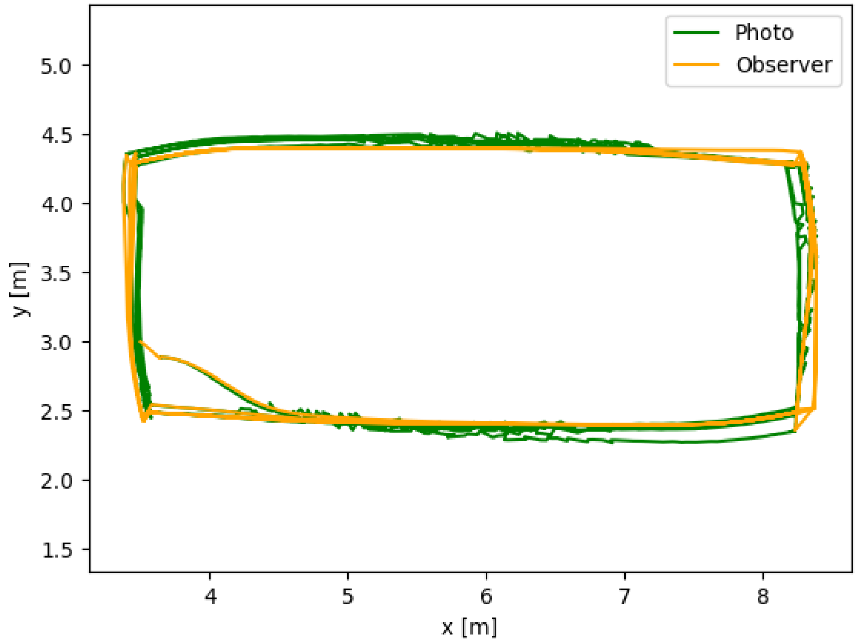

Figure 6.9: Position of the ATR when the controller is following the rectangle trajectory and using the look-and-move method.

Figure 6.10 shows the x and y position of the robot over time. For this case is it also clear that the observer position is corrected but the difference from the ellipse case is that the error is corrected in every corner when the robot stands still long enough for the observer to use the photogrammetry.

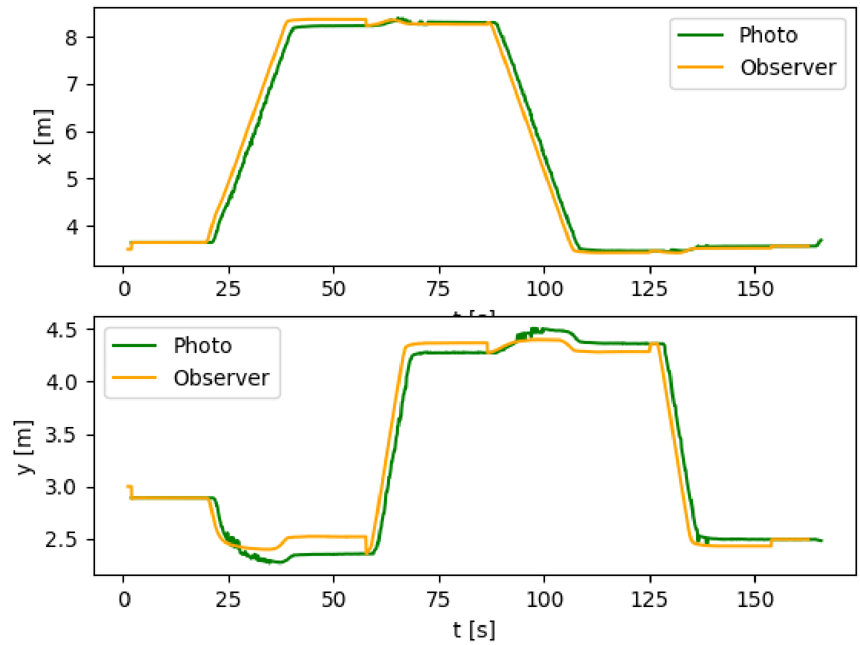

Figure 6.10: x and y positions of the ATR over time in the rectangle trajectory. The observer is using the look-and-move method.

6.4 Controller performance

The performance of the control system can be evaluated by looking at the data from the test cases where the controller is running with localization input from odometry, without correction from photogrammetry. Comparing the reference trajectory given to the MPC with the estimated state of the observer shows how well the control system can track the reference.

Figure 6.11 shows the result from the rectangle test case. The first two subplots shows the reference trajectory and observer estimate over time fot x and y respectivle. The third subplot shows the error in x and y but also the euclidean distance error for every time instance. It can be seen that the largest error is approximately $0.1m$, and it occurs in the acceleration phase after each corner.

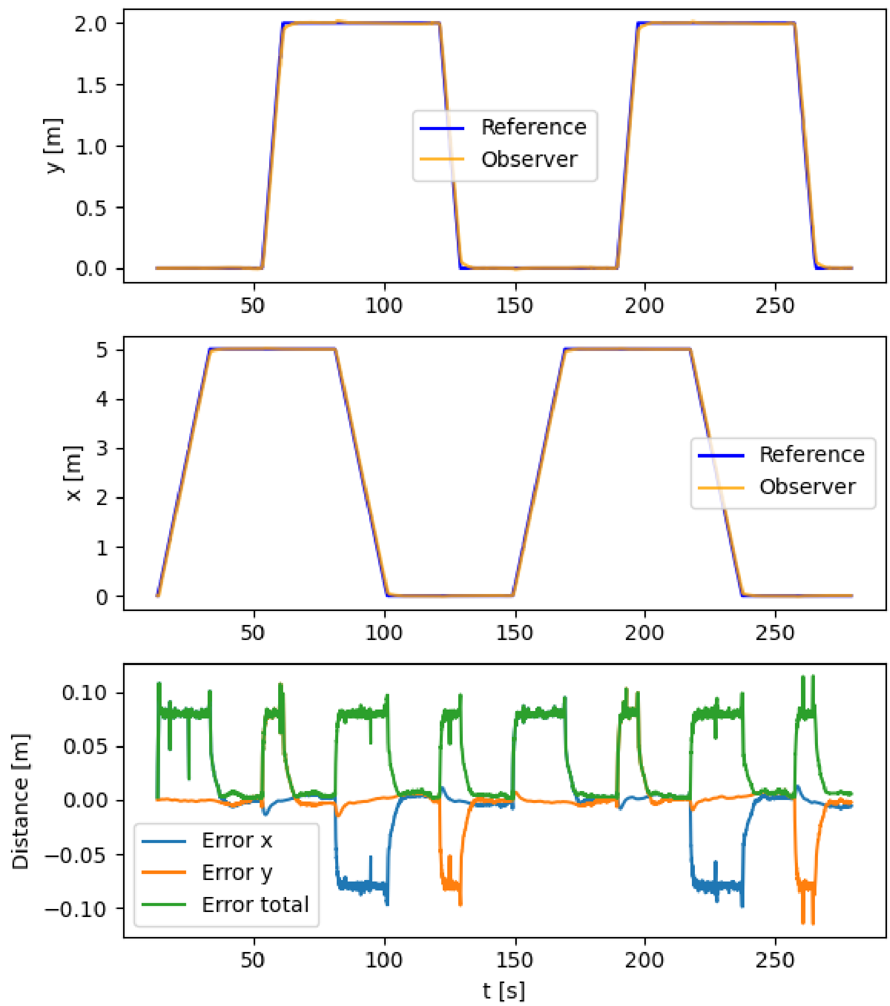

Figure 6.11: x and y-positions for observer and reference over time, and the error between observer and reference

Figure 6.12 shows the corresponding data from the test case with the ellipse-like reference trajectory. Also in this case is the largest error approximately $0.1m$.

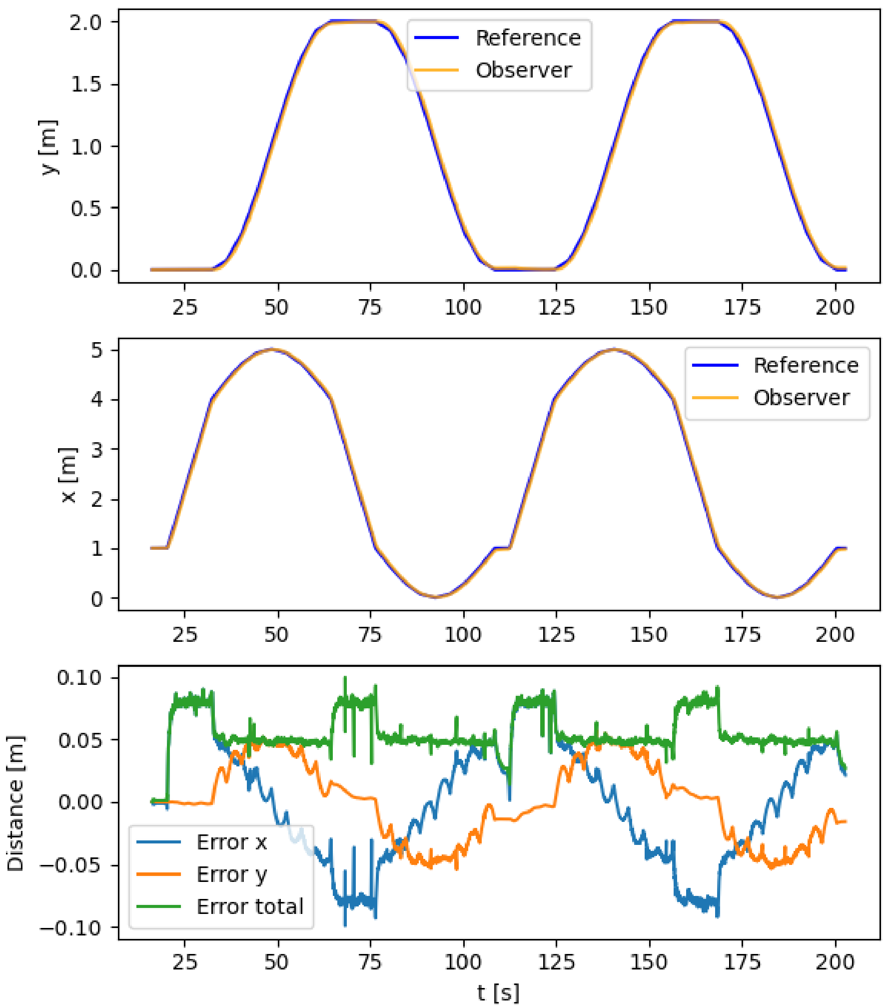

Figure 6.12: x and y-positions for observer and reference over time, and the error between observer and reference

6.5 Collision avoidance

The evaluation of the path-following system with reference input from the fleet control system indicates that collision between the two robots can be avoided. Since the fleet control system does not change the path of the ATR but only its positions in time, it may be noted from Figure 6.13 that the ATR takes shortcuts when it is not given enough time to take the sharp corners.

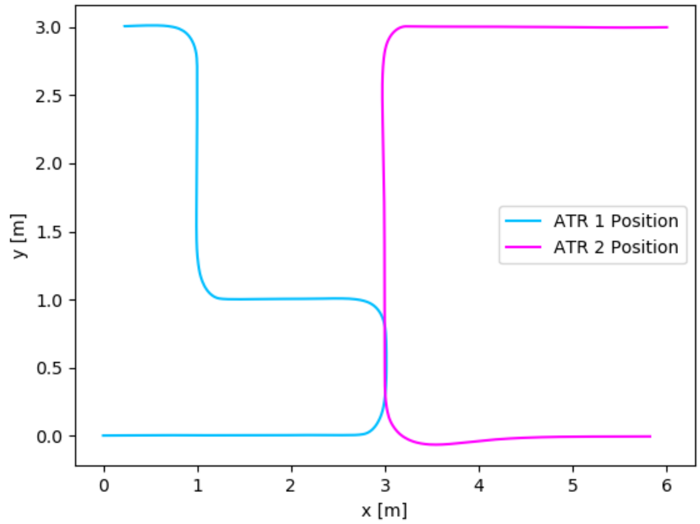

Figure 6.13: Position of the ATR estimated from odometry

Figure 6.14 shows the distance between the ATRs before changes made by the fleet control, during the run time measured by the observer, as well as the safety margin between the robots are depicted. The safety distance of 2 meters where uphold at all time.

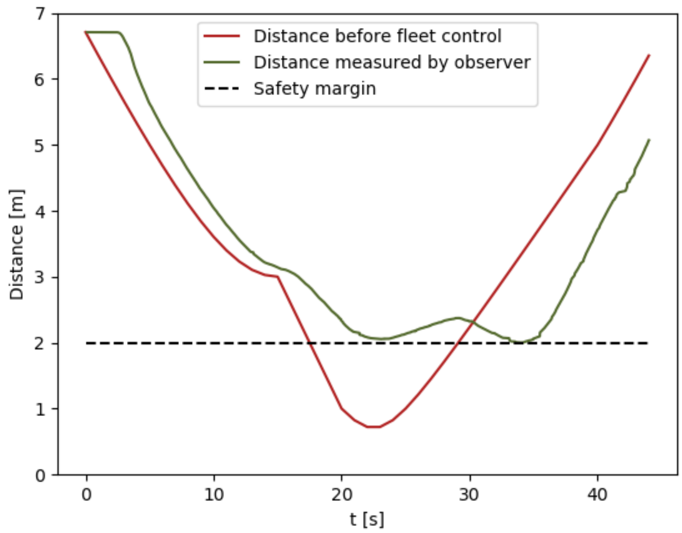

Figure 6.14: Distance between ATRs estimated from odometry

6.6 Computational efficiency

The temperature for the case where one instance of MPC was running can be seen in figure 6.15. The figure shows how the Raspberry Pi with the cooling solution holds a lower and steadier temperature throughout the entire test case than the one without cooling. Neither of the Raspberry Pis was close to 80 degrees Celsius, where the throttling is supposed to happen.

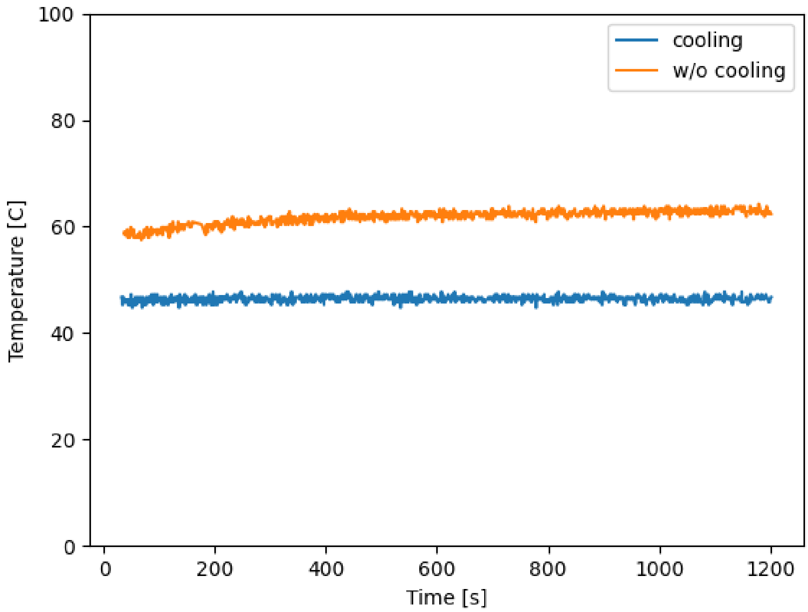

Figure 6.15: Temperature of the Raspberry Pi when running one instance of the controller

Figure 6.16 depicts the temperature for both Raspberry Pis when running three simultaneous instances of the MPC controller. The Raspberry Pi with the heatsink and fan runs at a similar temperature to what was seen in Figure 6.15 while running just one instance of the controller. In contrast, the temperature of the one without any cooling climbed steadily in the first half of the test and then flattened out for the remainder. This Raspberry Pi also reached temperatures of about 80 degrees Celsius.

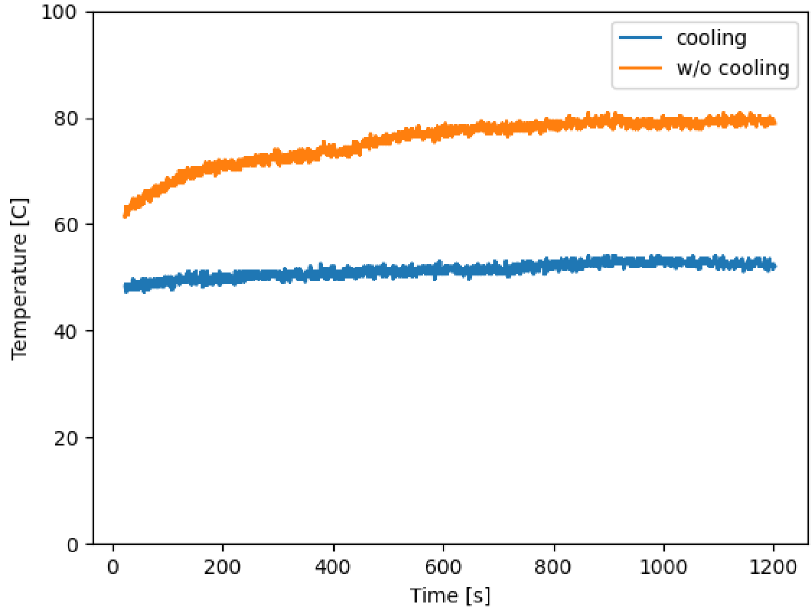

Figure 6.16: The temperature of the raspberry pi when running three instances of the controller

In Figure 6.17, the average MPC call frequency is shown, meaning how many times per second the ROS2 node is actuating the MPC to solve a new optimization problem. The figure shows how well Raspberry Pi performed, with and without the cooling. Both solutions operated around 90Hz during the entire 20 minutes of the test.

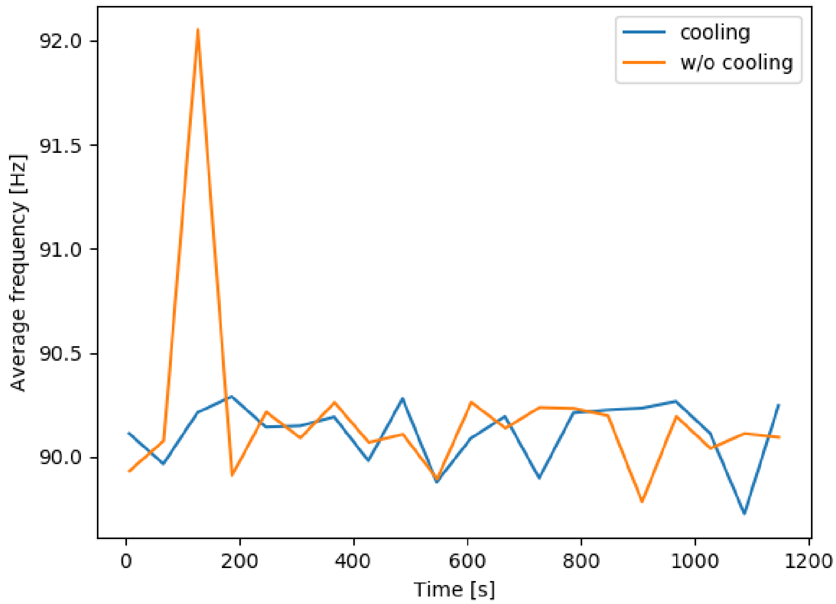

Figure 6.17: Average MPC call frequency for the test case running one instance of the MPC

In contrast to the test case above, the one depicted by Figure 6.18 shows a decline in average MPC call frequency during the test. The solution with the cooling system reached average frequencies as low as 58Hz, and for the Raspberry Pi without any such cooling frequency declined down to 50Hz

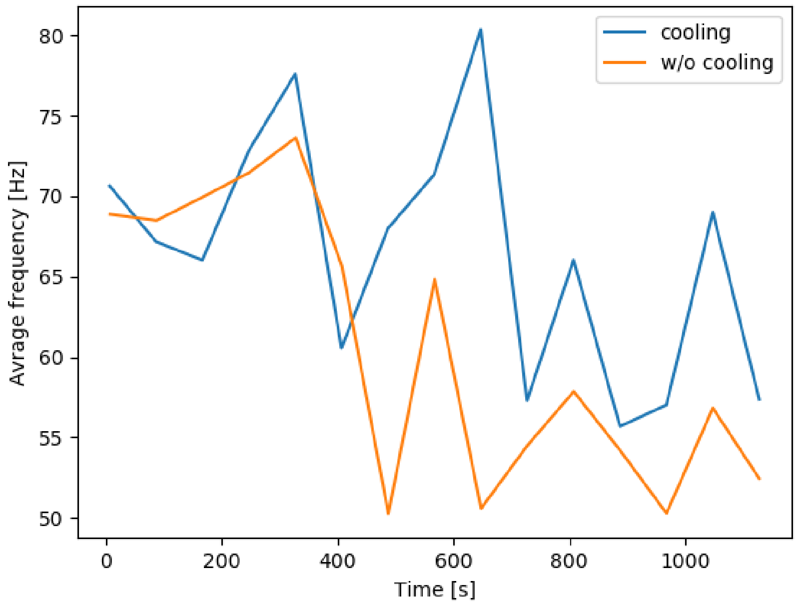

Figure 6.18: Average MPC call frequency for the test case running three instances of the controller

In Figure 6.19 the average solve time is detailed for the test case where only one instance of the MPC was running. The average solving time is mostly around 1 ms, with some spikes to around 3 ms.

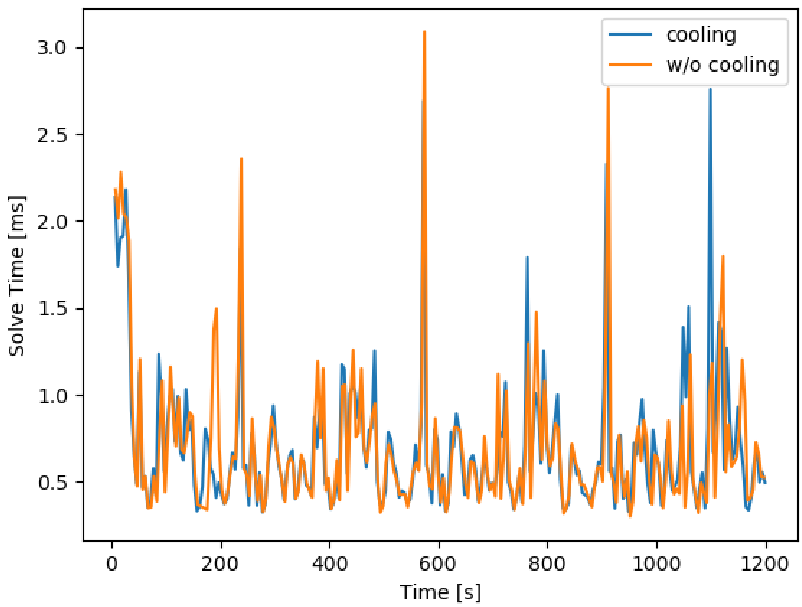

Figure 6.19: Average solve time for the MPC when running one instance of the controller

Figure 6.20 depicts the case where three instances of the MPC were running simultaneously. The characteristic for both the case with as well as without the cooling system is similar. The primary distinction between the cases is that the test without cooling reaches a higher average solve time at approximately 7.5 ms. In comparison, the test with cooling stayed lower, at around 5 ms.

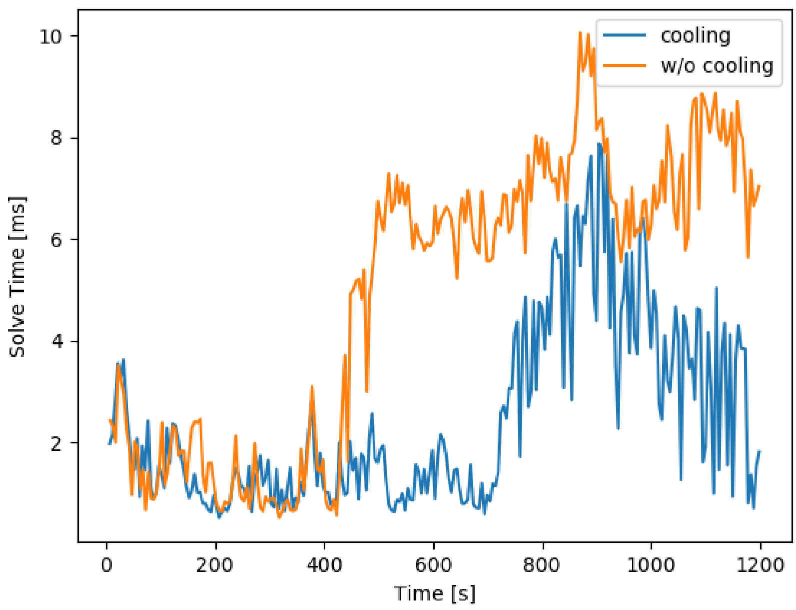

Figure 6.20: Average solve time for the MPC when running three instances of the controller

7

Discussion & Conclusions

Firstly, a very important aspect of the test results is that the embedded MPC-based path following control system is able to track the reference trajectory with a maximum error of 0.1 m. The test results also shows that this error is constant when driving at constant speed. One explanation for the error could be time delays in the system, caused by for example computational times, internal communication and actuators. Another explanation for the remaining error could be that the integral part of the PID controller was removed, making it, de facto, a PD controller. The tuning parameters of the MPC could also possibly be optimized to reach an even smaller error since currently its tuned for a smoother behaviour rather then an error minimizing one.

The results from the different test cases indicate that the odometry drift is primarily caused by the ATR turning. This means that the error is varying depending on the robot's reference trajectory. The photogrammetry correction is needed more often if the ATR is doing many turns and less if it is moving in straighter paths. This drift becomes substantial over time and this entails that the system will not be able to run entirely of the odometry. The photogrammetry, in turn, is very reliable in regards to its long-term accuracy. If the cameras has been calibrated well the photogrammetry provides an ATR position with a low margin of error. The photogrammetry does not need to be exact for the system to operate correctly since its primary use is to remove the odometry drift. Removing the drift can be done more seldom and then let the odometry provide short-term readings.

One drawback of the suggested localisation method with a Kalman filter is the fact that is it affected by potential latency in the vision system. If this delay is to large, the photogrammetry will cause more error and instability rather than correcting the position. The current solution, the look and move approach, removes this problem, but can instead potentially make the whole delivery system much slower, if the robots needs to stop and wait at non-desired locations. A future solution could be to use a camera system with a smaller latency. If this is not possible, another solution is to take the delay into account in the observer by fusing the photogrammetry data with odometry from the moment backwards in time corresponding to the latency. This method requires knowledge about how big the delay is, either if it is constant and can be measured or by providing the information to the observer together with the photogrammetry data at every measurement.

Putting together the tracking error from the controller together with error in localization gives information about how large the safety margins need to be to avoid collisions. In the greater perspective, the delivery system has to have safety margins around each ATR and obstacle and this will affect the efficiency of the system. The magnitude of these safety margins will be determined by the accuracy of the path following system. There will always be errors in both reference tracking and localization and the larger these errors are, the larger safety margins are needed. Using the look and move method may also introduce some lag in the system, which can lead to a slower system that can transport fewer goods.

It is clear from the test case based on the trajectory from the fleet control module that the path following system works well when it comes to collision avoidance between robots that are controlled by the fleet control system. However, collision avoidance with other dynamical objects may require another behavior and put other demands on the system. The fleet control system does only change the trajectory in time and not in space, while the collision avoidance system can change the trajectory also in space.

Further, when evaluating the solve time of the MPC it seems like running the control system on an embedded single-board computer is feasible. The average solving time is below 10 ms even during high load situations. It may be noted that the difference in performance between the Raspberry Pi that used the cooling system in comparison to the one that did not is quite substantial during the high loads. The instance without cooling had solve times that were around 50 percent higher during high loads. It should also be noted that the average frequency the MPC was able to run at was affected by both cases. Using cooling was better but not with the same margin as the average solve times.

The current implementation with different ROS2 nodes for internal communication was chosen to create modularity in the system. On the other hand, the control system may be faster and able to run at a higher frequency if only using one ROS2 node, but at the cost of the modularity.

Conclusion

The odometry drift is substantial enough to need photogrammetry to remove the drift when it becomes too large. Since there is a considerable delay in the photogrammetry, this leads to the need for the look and move method to let the photogrammetry data catch up to the system. Using the look and move method with the observer, reset the drift caused by odometry after each lap during testing. When testing how well the controller manages to follow the reference trajectory, it reached a maximum distance error of 0.1 meters. The possible highest frequency of the path-following system was 90 Hz during the trajectory with added complexity.

Bibliography

[1] J. Andersson, J. Åkesson, and M. Diehl. Dynamic optimization with casadi. In *2012 IEEE 51st IEEE Conference on Decision and Control (CDC)*, pages 681–686, 2012.

[2] T. Benoit-Cattin, D. Velasco-Montero, and J. Fernández-Berni. Impact of thermal throttling on long-term visual inference in a cpu-based edge device. *Electronics*, 9(12), 2020.

[3] B. Bischoff, D. Nguyen-Tuong, F. Streichert, M. Ewert, and A. Knoll. Fusing vision and odometry for accurate indoor robot localization. pages 347–352, 12 2012.

[4] J. Borenstein and L. Feng. Measurement and correction of systematic odometry errors in mobile robots. *IEEE Transactions on Robotics and Automation*, 12(6):869–880, 1996.

[5] A. Cavin. Obstacle avoidance for mobile robots using a photogrammetry-based sensor system. Master's thesis, Swiss Federal Institute of Technology Lausanne, 2019.

[6] B. Egardt. *SSY208 Model Predictive Control, Lecture Notes*. Signals and Systems Chalmers University of Technology, 2020.

[7] M. Kamel, M. Burri, and R. Siegwart. Linear vs nonlinear mpc for trajectory tracking applied to rotary wing micro aerial vehicles. *IFAC-PapersOnLine*, 50(1):3463–3469, 2017. 20th IFAC World Congress.

[8] K. C. J. F. T. F. J. L. E. B. R. W. M. Quigley, B. Gerkey and A. Ng. Ros: an open-source robot operating system. page 6, 2009.

[9] D. Merkel. Docker: lightweight linux containers for consistent development and deployment. *Linux journal*, 2014(239):2, 2014.

[10] I. Nigri, M. A. Meggiolaro, and R. Q. Feitosa. Comparison between look-and-move and visual servo control using sift transforms in eye-in-hand manipulator system. In *20th International Congress of Mechanical Engineering*, 2009.

[11] K. Pålsson and E. Svärling. Nonlinear model predictive control for constant distance between autonomous transport robots. Master's thesis, Chalmers University of Technology, 2020.

[12] A. Sathya, P. Sopasakis, R. Van Parys, A. Themelis, G. Pipeleers, and P. Patrinos. Embedded nonlinear model predictive control for obstacle avoidance using panoc. In *2018 European Control Conference (ECC)*, pages 1523–1528, 2018.

[13] P. Sopasakis, E. Fresk, and P. Patrinos. OpEn: Code generation for embedded nonconvex optimization. In *IFAC World Congress*, Berlin, Germany, 2020.

[14] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos. A simple and efficient algorithm for nonlinear model predictive control. 09 2017.

[15] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos. A simple and efficient algorithm for nonlinear model predictive control, 2017.

[16] K. J. Åström and T. Hägglund. *PID Controllers: Theory, Design, and Tuning.* ISA - The Instrumentation, Systems and Automation Society, 1995.

DEPARTMENT OF SOME SUBJECT OR TECHNOLOGY
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden
www.chalmers.se

AVANCEZ
1829

CHALMERS
UNIVERSITY OF TECHNOLOGY