Autonomous driving using Model Predictive Control methods

Johan Kellerth Fredlund

Kenan Sadik Sulejmanovic

LUND UNIVERSITY

Department of Automatic Control

MSc Thesis
TFRT-6029
ISSN 0280-5316

Department of Automatic Control
Lund University
Box 118
SE-221 00 LUND
Sweden

Abstract

This thesis explored the path following applications of autonomous driving, where the purpose is to sense the environment and navigate to a goal without human intervention. Autonomous driving enables safer journeys by removing human error in driving, as well as reducing fuel consumption and driving time by optimizing the engine and brake actuation. In 2017 it is a well researched topic in the academic and industrial world.

The scope of this work was to create a reliable and efficient path following solution which enabled a robot car to accurately traverse along any given path. The objective was therefore to minimize path error with the additional objective to minimize the traversal time.

The method used to realize the objective statement was Model Predictive Contouring Control (MPCC). MPCC uses optimization based prediction to smoothly control the car along a reference path, where actuator constraints and the objective are both handled in the optimization problem. The reference was also defined as a geometric function instead of a specified trajectory of coordinates. With the addition of objective weights it was possible to choose the relative importance of the two objectives, path accuracy and minimizing time. The resulting problem formulation, which was heavily non-linear, was linearized to reduce computation time and solved using FORCES Pro. Simulations where made in Simulink, while real-time execution of the MPCC were achieved with a robot prototype, Robotics Operating System (ROS) and Matlab.

The result suggested that MPCC was able to operate efficiently in a real time environment. Manipulating the objective weights resulted in predictable operation, which makes it possible for a user to control whether faster traversal or tracking performance is desired. It can be concluded that MPCC is a viable control method for autonomous driving, in particular for autonomous racing purposes. Additionally, the simulation results regarding the non-linear MPCC showed a surprisingly fast solve time, which suggest that it would be feasible for on-line driving using the solver FORCES Pro.

Acknowledgements

First of all we would like to thank our supervisors Anders Robertsson and Bo Bernhardsson for their support during the thesis, for giving us helpful suggestions and providing us with the prototype that we used for the real time experiments. We would also like to thank our families and friends who have supported us during our master thesis and LTH education. Specifically, a big thanks to Santiago Castro and Koshin Aliabase for being great friends and working partners throughout our time at LTH. We would also like to thank Embotech, for letting us use the high speed solver FORCES Pro, and their support in dealing with queries regarding implementation. Finally we thank Uniti for letting us do our master thesis at their company and our assisting supervisor Michael Bano for his support.

Contents

1 Introduction

1.1 Background

Autonomy in vehicles is a rapidly expanding technology that is of interest in many major car companies such as Volvo, Tesla and Mercedes [24]. This interest in autonomous driving is not exclusive to car companies, however, as Scania is developing safe and realiable self-driving trucks that can be used in mines [21]. While certain autonomous functions in cars have been in use for over a decade, such as lane keeping and automatic parking these features are subject to specific scenarios, a fully automated vehicle needs to take into account the unpredictable and complex environments that cars drive in. That is why with current technology autonomous cars are mainly planned to be used on larger roads. The idea of a self-driving car in a city environment needs specific requirements such as reliable path following, accurate sensor data and safe driving.

1.2 Problem formulation and motivation

In this thesis the theory of optimization will be utilized to efficiently control a car on-line. By using predictive control based optimization with respect to a model the control possibilities are numerous. The main goal is to design a smooth and reliable path following algorithm with the focusing objective of path tracking while maximizing speed, given any generated path trajectory. The design should also support expansion of the objective in any desired way. That can be to reduce fuel consumption or reducing jerk.

This thesis work has been in cooperation with Uniti, a company that is making an electric car intended for city driving. Our contribution comes in the form of creating parts of the autonomous features intended for their vehicle.

1.3 Thesis scope

To reduce the scope of the project path following aspect of autonomous driving is the exclusive focus, which makes it possible for extensive research of an optimal control method. A kinematic model for the car was used to reduce the scope of the project and because at low speeds a kinematic model is sufficiently accurate [13].

2

Theoretical Background

This chapter will explain the most important theory behind the path following methods. The theory behind Model Predictive Control will be explored, as well as the theory behind optimization, the solver used, and the position estimator used during real-time experiments.

2.1 Model Predictive Control

Model Predictive Control (MPC) is an advanced control method that works in discrete time. From a set of state values, and with respect to a model, it optimizes a problem around an objective and gives a sequence of control signals as outputs. The first set of control values are then used as inputs to the system plant, and after a short period, set as the system time step, the new state values are measured and the process is repeated. In this section we will shortly describe the history of MPC and give some basic examples of its structure and the theory behind it.

History

The beginning of MPC was at Shell Oil Company in 1979 where an idea named as "Dynamic Matrix Control" was presented by Cutler and Ramaker ([5], [17]). DMC was the first type of predictive control that could be applied in industry. The idea was to handle multivariable control systems without any constraints and predict future values for linear systems. The idea that the algorithm would predict future plant behavior was discovered to lead to a less aggressive output and a smoother convergence to the target set point. Throughout the 80s MPC was popular mainly in industries such as chemical plants and oil refineries, i.e. in slow processes where the computational time of the solvers would not be a problem. In the 90s the theory of MPC matured and with faster solvers and computers the algorithm was now feasible for faster, more demanding systems. Today MPC has many applications, and as we will demonstrate in this thesis, one of them is in autonomous driving.

Predictive Control

The prediction horizon Prediction is at heart a human capability, and therefore it might be more suitable to call the invention of predictive control a discovery. When we are driving we never look straight down at the road, but farther ahead. The reason is of course so that we can plan our driving. When a sharp turn approaches we need to brake ahead of time. A driver always looks far enough to ensure safe driving, so called minimum braking distance, in case of an unexpected obstacle on the road. This should also apply in control. In predictive control there is a finite prediction horizon set for each optimization, i.e., how far the controller looks into the future. For an MPC applied in autonomous drive the algorithm predicts the motion of the car, with respect to a model, and decides its control signals to follow the desired path. This leads in essence to safer driving and better handling of unexpected disturbances and obstacles. To decide the length of the horizon we can again draw an analogy to human driving. While driving at high speeds you need a longer prediction horizon since the minimum breaking distance is also longer. Looking at Figure (2.1) the prediction horizon must be long enough such that distance between the two cars are larger than the minimum braking distance. A longer horizon is usually ideal but is often limited by sensor limitations [19]. The computational complexity also increases for longer horizons, mainly for complex non-linear systems.

Figure 2.1: Human prediction in driving [6]

Another analogy can be seen in chess. The opponent can be seen as a relatively predictable disturbance in the game. You need to predict the moves of the opponent several steps ahead in order to win, a long horizon and a good disturbance model will increase your chance to win.

Concept of prediction In MPC discrete-time models are used. Using the current state values we predict one step ahead with a time step h.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ x_{k+2} &= Ax_{k+1} + Bu_{k+1} \\ &\vdots \\ x_{k+H} &= Ax_{k+H-1} + Bu_{k+H-1} \end{aligned}$$

where H is the prediction horizon. This one-step prediction can then be used recursively to find the n-step prediction by inserting the expression of x_{k+1} in x_{k+2}, x_{k+2} in x_{k+3} etc.

$$\begin{aligned} x_{k+1} &= Ax_k + Bu_k \\ x_{k+2} &= A^2x_k + ABu_k + Bu_{k+1} \\ &\vdots \\ x_{k+H} &= A^Hx_k + A^{H-1}Bu_k + A^{H-2}Bu_{k+1} + \cdots + ABu_{k+H-2} + Bu_{k+H-1} \end{aligned}$$

The predictions at time k is then expressed as

$$\begin{bmatrix} x_{k+1|k} \\ x_{k+2|k} \\ \vdots \\ x_{k+H|k} \end{bmatrix} = \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^H \end{bmatrix} x_k + \begin{bmatrix} B & 0 & \dots & 0 \\ AB & B & \dots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ A^{H-1}B & A^{H-2}B & \dots & B \end{bmatrix} \begin{bmatrix} u_{k|k} \\ u_{k+1|k} \\ \vdots \\ u_{k+H-1|k} \end{bmatrix} \tag{2.1}$$

where $x_{k+1|k}$ is the one-step prediction at time k.

Intuitively it can be seen that the prediction structure is built such that the decision variables $u_{k|k}$ ... $u_{k+H-1|k}$ controls the state trajectory, which is important in trajectory tracking when we often desire specific predictions.

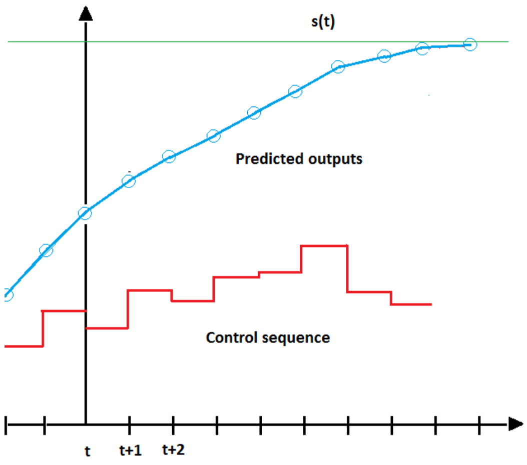


Figure 2.2: Predictive control for a system with $H = 9$ and with a control objective to reach a set point $s(t)$

Figure (2.2) shows a typical example of predictive control where the goal is to reach a certain set point $s(t)$. The decision variables correspond to a predicted output, and the control output at time $t{+}1$ will be supplemented to the plant. If the model is good enough the predicted output at time $t{+}1$ will be close to the measured value.

MPC problem structure

In most cases the model is a linear approximation of the actual model but non-linear models can also be used in an MPC formulation. Non-linear models require specific solvers that can perform computations of non-linear optimization problems. In this thesis we will use a linear model

$$x_{k+1} = Ax_k + Bu_k \tag{2.2}$$

$$x_k \in \mathbb{R}^n, \; u_k \in \mathbb{R}^m \tag{2.3}$$

where m indicates the degrees of actuation of the system. In most systems there are actuator constraints as well as state limitations. In common control systems there is a risk of saturation since these constraints are not explicitly taken into account during the control procedure. Fortunately in MPC all constraints are handled as a natural part of the problem, as long as they are convex. The constraints are expressed as linear inequalities [19], [12], [1].

Both of the inequalities form a polyhedron, which is a convex set, Figure (2.3)

$$Fu_k \leqslant b_u \tag{2.4}$$

$$Gx_k \leqslant b_x \tag{2.5}$$

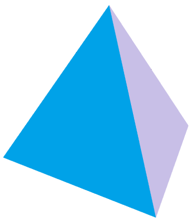


Figure 2.3: A visualization of a polyhedron, a convex set and a typical form of a feasible set in $\mathbb{R}^3$

The problem objective is expressed as a quadratic program. A basic problem formulation looks like

$$\begin{aligned} \text{minimize} \quad & \sum_{k=1}^{H} x^T Q\, x + \Delta u^T R\, \Delta u \\ \text{subject to} \quad & x_{k+1} = Ax_k + Bu_k, k = 1, ...n, x \in \mathbb{R}^n, u \in \mathbb{R}^m \\ & u_k \in \mathbb{U} \\ & x_k \in \mathbb{X} \\ & g(u_k, x_k) \leqslant 0 \end{aligned} \tag{2.6}$$

where the objective is minimized over the horizon H with respect to the model, the variable constraints and possibly other system or control limitations. Variable values outside of the constraints will therefore not be mathematically feasible, which is preferable in cases with system instabilities outside of its hard constraints. An MPC works in stages and using the model it predicts plant behavior several steps in the future which for accurate models leads to predictable behavior. The objective is subject to several iterations depending on the horizon set for the optimization. The resulting output of the optimization is a sequence of control signals and predictions.

Model Predictive Control works with the receding-horizon idea. For each time step it optimizes the objective function around a finite time horizon, computing a set of control signals u_1,....,u_n that solve its problem objective with respect to the set constraints. In path following applications the output control trajectory will correspond to predictions of the vehicles position at a given time step. Only the first control signals u_1 will be sent to the plant and the controller will perform another optimization, where a new control sequence is calculated. In the new optimization the horizon has therefore moved one step into the future and the horizon has moved.

Algorithm 1 Basic MPC control loop

```
1: Measure $x_k$
2: Solve the problem formulation (2.7) with $x_k$ as initial state, where $u_k$ is
   calculated
3: Set $u_k$ as input to the plant
4: Repeat steps 1-3
```

An MPC has many strengths. Given that the model is discrete and linear it handles multivariable problems very well. This is very much dependent on whether the resulting problem formulation is convex or not, which will be discussed in the next section. The optimization handles actuator constraints and state constraints naturally in the optimization which allows for the process to be operated much closer to the hard constraints, which improves control performance and efficiency. Because of its predictive nature it is able to solve a variety of problems and handle disturbances smoothly.

2.2 Convexity

Mathematical convexity is an important part of the problem formulation of an MPC. Convexity is a mathematical property given to a range of functions and sets. A set is convex if any two points that lies on the set can be connected by a line which completely falls inside the set [2], [1]. This can be seen in Figure (2.4).

$$
\begin{aligned}
&C \text{ is a convex set.}\\
&\text{then for all } x,\ y\ \in\ C,\\
&\theta x\ +\ (1-\theta)y\ \in\ C,\\
&\text{for all } 0\ \leqslant\ \theta\ \leqslant\ 1
\end{aligned}
$$

Convex sets are essential in mathematical optimization to avoid non feasible solutions. In convex optimization the variable sets need to be convex in

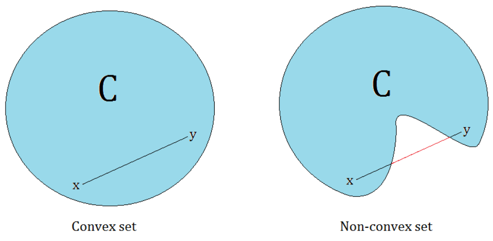


Figure 2.4: Convex sets visualized.

order for the solver to be guaranteed to find an optimal solution. If you set a non-convex constraint of an optimization variable the resulting objective function will be subject to a feasible domain that is not a convex set.

A convex function is a function with a domain that is a convex set and has a convex shape. That means that the straight line between two points on a convex curve is at every point over or equal to the function (Figure 2.5). Another definition is that the epigraph of the function is a convex set. The mathematical definition of a convex function is:

f is a function with the domain C.

It is a convex function if:

for all x, $y \in C$,

$f(\theta x + (1-\theta)y) \leqslant \theta f(x) + (1-\theta)f(y)$

for all $0 \leqslant \theta \leqslant 1$

Convexity is important in any type of mathematical optimization, and even more for applications intended for on-line computations. For an MPC optimization problem there will be a set time-step before the problem needs to be computed again with possibly new state values. A convex function has only one minimum and is therefore preferable for on-line execution. An optimization problem is described below.

$$\begin{aligned} \text{minimize} \quad & f_0(x) \\ \text{subject to} \quad & f_i(x) \leqslant 0, i = 1, ..., m,\ x \in \mathbb{R}^n, u \in \mathbb{R}^m \\ & h_k(x) = 0, k = 1, ..., n \end{aligned} \tag{2.7}$$

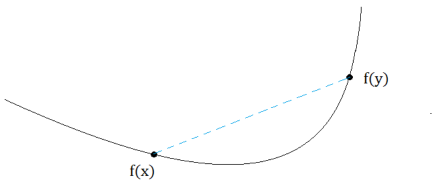


Figure 2.5: A depiction of a convex function. Notice that the line segment between the points $f(x)$, $f(y)$ are above the function at all times.

For it to be a convex optimization problem $f_0(x)$ needs to be a convex function and the domain of x needs to be a convex set. The set of constraints $f_i(x)$ all need to be convex functions and the equality constraints $h_k(x)$ must be affine. An affine function is also convex. The definition of an affine function is:

$$\begin{aligned}
&f \text{ is affine if it is convex and}\\
&\text{for all } x,\ y\ \in\ C,\\
&f(\theta x + (1-\theta)y)\ =\ \theta f(x) + (1-\theta)f(y)\\
&\text{for all } 0\ \leqslant\ \theta\ \leqslant\ 1
\end{aligned}$$

If you can prove that a problem formulation is convex then you can guarantee convergence to a global minimum in a predictable time [2]. Non-convex optimization problems and constraints can also be used for MPC purposes. Dynamic models are non-linear and must be linearized around a point or a trajectory before it is affine. Using a non-linear model will give more accurate horizon estimates, but will make the problem more computationally complex. There are, however, solver methods for non-linear optimization problems which will be mentioned in the next section.

Proving convexity can be a difficult task. Visually it is easy to see whether a function is convex or not in a 2-D or 3-D graph. However, many optimization problems have more than two variables which makes visualization difficult.

That is why in most MPC problems the problem formulation is expressed as a Quadratic Program (QP). A standard QP is $x^T Qx + cx$. If Q is positive semi definite the function will have an upward curvature for x in $\mathbb{R}^N$. In linear programming $Q = 0$ which will give an affine problem formulation, which is also convex.

In other optimization problem which are not QP techniques have to be used to prove convexity. One of those techniques is to prove upward curvature by calculating the Hessian matrix of the function. If the Hessian matrix is positive semi definite then the function is convex.

2.3 Solvers

There are several optimization methods that can be used and solvers are typically designed to use more than one method for different purposes. The solver that we use is FORCES Pro [3]. FORCES Pro is a high speed solver intended for real time execution. It uses the solver method Interior Point Method [4] as the standard solver method, usable for both convex and non convex optimization problems. In this thesis we will take full advantage of the solver and implement both convex and non convex problems.

2.4 Simultaneous Localization And Mapping

SLAM is short for simultaneous localization and mapping. Rao-Blackwellized particle filter is one technique that produces a map and a location in the map. In Robotic Operating System (see Chapter 3), the package name using this filter is called `Gmapping`, which has been utilized to create the map used in the experiments. For further details on the filter see [9].

2.5 Probabilistic Robotics

"Robotics is the science of perceiving and manipulating the physical world through computer-based mechanical devices" [23]. A key element of robotics is uncertainty. Uncertainty emerge if the robot lacks critical information for executing out its task, e.g., when implementing a control method in a robot the position and the angle of the robot are required.

Uncertainty

Uncertainty can come from different sources, such as sensors, the robot and the model. The sensors are subject to noise, resulting in unpredictable sensor measurement. This fact limits the information that can be extracted. Robot movement requires motors, which can be slightly unpredictable due to the effects like control noise and wear-and-tear. The model in our case is an estimation of the dynamics of the vehicle which leads to model error, i.e., another source of uncertainty. In real-time, there is a limitation of how much computations that can be carried out.

Estimating the states with probability theory

As mentioned before, the position and angle estimation is required for this work. Therefore the localization of the robot is needed. Localization is the problem of estimating a robot's coordinates in an external reference frame from sensor data, using a map of the environment. In this work a probabilistic localization method is used. The key idea of probabilistic robotics is to represent uncertainties explicitly. This is done by using probability theory. Probabilistic algorithms represent information by probability distributions over all possible hypotheses.

The Bayesian approach is one way of solving the problem of estimating the position and angle of the robot. In general the Bayesian inference method is a method where the probability is updated as more evidence or information becomes available. The method is build upon Bayes' theorem, see equation 2.8. The solution of equation 2.8 is called the posterior distribution.

$$P(A|B) = \frac{P(A)\,P(B|A)}{P(B)} \tag{2.8}$$

Bayes filter

Bayes filter is the most general algorithm for calculating the belief. In this work the belief is the three states x and y coordinate and the angle in the map. Bayes filter is recursive and contains two core steps, the prediction and measurement update.

The first step is called the prediction or the control update. At this stage the state x for time $t-1$, control signal for time $t-1$ and time t is known. Notice that the control signals are not what they could have been in a traditional sense. The control signals in this case are the local states of the robot in a local coordinate system for time $t-1$ and t.

The second step is called the measurement update. The belief of the previous step is multiplied with the probability that the measurement z has been observed at time t given the state x. This is repeated for each hypothetical posterior state x for time t.

There is a parametric and non parametric Bayes filter. In this work we have used a nonparametric filters because the posterior can be represented by finite samples. An additional advantage of the nonparametric filter is that it can cope with uncertainty from sensors and multiple possible hypotheses.

Particle filter

The specific nonparametric filter which was used in this work is the particle filter. The particle filter is a implementation of the nonparametric Bayes filter. The posterior probability is represented by particles, i.e., a set of hypotheses at time t, where the distribution of the set determines the posterior distribution. Each particle is a state of what the true world state may be at time t, e.g., the position of a robot in a given map.

Adaptive Monte Carlo Localization

Adaptive Monte Carlo Localization (AMCL) is an adaptive particle filter.

In the case of the pose of the robot, one particle is one assumption out of many in the posterior set of assumptions. All the particles together make up the distribution of the believed poses for the current time instance, where these particles represent a set of beliefs of the true state. Furthermore, a dense sub-region of the state space increases the probability that the true state falls into that region, according to [23], in Chapter 4. This representation is an approximation of the distribution of the state. It can therefore represent a much broader space of distributions than, for instance, a Gaussian distribution.

The general structure of the AMCL filter will now be addressed (Figure 2.10). When initialized, a set of random particles are produced, alternatively they are set by an external source. In Step 1, new beliefs are generated according to a probabilistic motion model using odometry, see Figure (2.8). This step is the prediction step mentioned in the Bayes filter section. In this work the rotary sensors on the wheels of the robot was used to calculate the distance the robot has traveled. But since there is slip and other disturbances the distance traveled will not be accurate. In addition, studying the pseudocode `sample_motion_model` from [23], the control signal is a set of measurements from the odometry.

Moreover, the movement from any point A to any point B by the wheeled robot can be represented by a sequence of three independent movements. A rotation, followed by a straight line motion and another rotation (Figure 2.6). The translation and the turns are noisy, the parameters α_1 to α_4 scales the variance of the noise independently for each δ_i (Figure 2.6). The noise models the errors due to slip, drift and faulty measurements. The distribution of the posterior poses vary for different sets of α_1 to α_4 values (Figure 4.1).

It follows in Step 2 that a measurement is taken from the sensor, which goes to the measurement model along with the newly generated state $\mathbf{X}$, see Figure (2.9). Hence, we get a weight $\mathbf{W}_i$. This step is the measurement update mentioned in section Bayes filter.

The weight represents the probability that we are at the state $\mathbf{X}_i$ and simultaneously have the actual measurement $\mathbf{M}_i$ by the real sensor, e.g., a Light Detection and Ranging sensor (LIDAR).

Using the model of the `likelihood_field_range_finder_model` given in [23], we compute the probability of the measurement given a pose of the

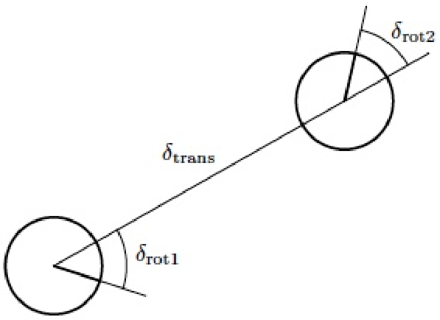


Figure 2.6: The movement from any point A to any point B can be described in three separate movements. A rotation δ_{rot1}, translation δ_{trans} and a final rotation δ_{rot2}. Figure taken from [23].

.

Parameter	Expected variance scaling in noise when the odometry does a;	When the robot does a;
α_1	rotation estimation	rotation
α_2	rotation estimation	translation
α_3	translation estimation	translation
α_4	translation estimation	rotation

Table 2.1: The specific interpretation of the $\alpha_1 - \alpha_4$ values which models the variance scaling in noise. These noises are assumed to be independent of the three virtual movements, δ_{rot1}, δ_{trans} and δ_{rot2}. See Figure (2.6) for the three virtual movements which represent all the movements done when traveling from any point A to any point B.

robot, i.e., the weight in Figure (2.9) and (2.10). Two parameters can be set in this procedure, z_{hit} and z_{rand}, which reflects how probable the measurements are from the sensor, e.g., a LIDAR.

It follows that we save every proposed state with the corresponding weight in a temporary set, see Figure (2.10) Step 3. Step 1 to Step 3 is repeated until all of the old particles have been renewed.

Turning to Step 4, there are two options. One that generates a random pose and the other that draws a pose from the set B with probability $\mathbf{W}_i$. The first to be considered is Step 4 a) and then 4 b). Now in Step 5 the chosen pose is then set into the set A, which represents the new set of particles

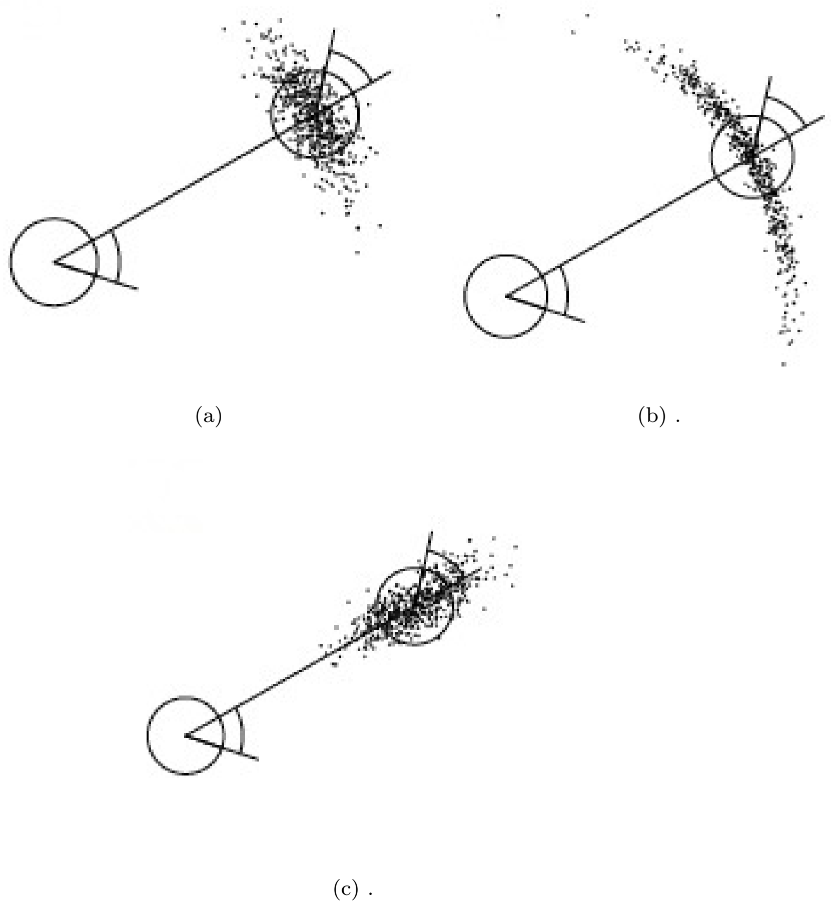


Figure 2.7: Three set of different set of α_1– α_4 noise parameters, where the dots represents the sampled distribution of probability where the robot will end up. Figure taken from [23].

(hypotheses of the new pose). Moreover if the random pose is chosen, the drawing from the set B will not be considered for that loop instance. The loop from Step 4 to 5 exits when the number of poses, in set B, is according to the KDL-Sampling algorithm, [22].

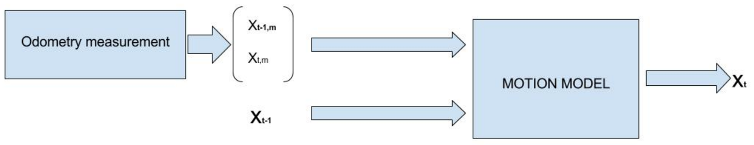


Figure 2.8: Step 1 in Figure (2.10) which is the prediction in Bayes filter. Notice that the new position in the local coordinate system of the robot is known, however the position in the global coordinate system is searched for. In this sub-system the movement of the robot from any point A to B is represented in three independent movements, δ_{rot1}, δ_{trans} and δ_{rot2} with independent noises. $\alpha_1 - \alpha_4$ represent these noises, see Table (2.1), Figure (2.6) and (4.1).

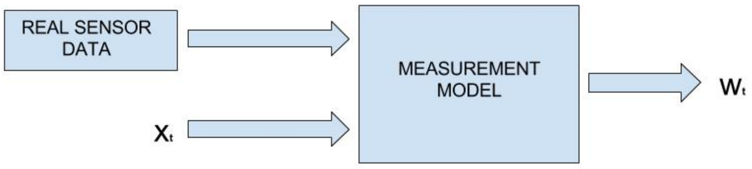


Figure 2.9: Step 2 in Figure (2.10) which is the measurement update in Bayes filter. Real sensor data is collected and the hypothesised pose $\mathbf{X}_t$ of the robot is accepted as input. It follows that the output is a weight which represents how probable the pose together with the real measurement is.

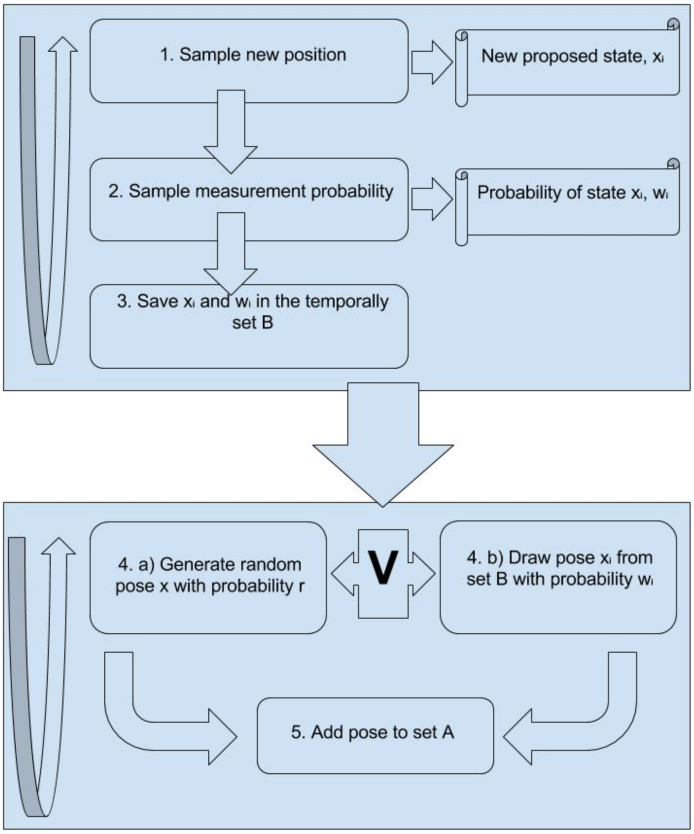

1. Sample new position
New proposed state, xi
2. Sample measurement probability
Probability of state xi, wi
3. Save xi and wi in the temporally set B
4. a) Generate random pose x with probability r
V
4. b) Draw pose xi from set B with probability wi
5. Add pose to set A

Figure 2.10: Simplified overall structure of MCL, [22] and pseudocode `Augmented_MCL` from [23].
Step 1: A new pose is sampled using an old hypothesized pose, the old and the new pose in the local coordinate system of the robot. See Figure (2.8) for further details.
Step 2: The probability of the hypothesized pose sampled in Step 1 and the real-time sensor measurement is the output in this Step. See Figure (2.9) for further details.
Step 3: The pose sampled in Step 1 and the probability-weight w is put in a temporally set B.
Step 4: Either a random pose $\mathbf{X}_t$ is generated or a pose from set B with a probability of W_i is drawn.
Step 5: The chosen pose from Step 4 is inserted in the set A.

2.6 Vehicle Modeling

The differential drive system is a robot with two wheels, mounted on a common axis (Figure 2.11). Turning is achieved by having different speeds on the wheels. For moving in a straight line both wheels therefore have the same speed.

The center of the robot is defined by the local coordinate system, with axis x and y (Figure 2.11). While the global coordinate system axis are X and Y, e.g., the position in a map is in the global coordinate system with the axis X and Y.

The differential drive model, Equation (2.9), is a kinematic model of the prototype. The control u_1 and u_2, is the linear respectively the angular speed of the robot. θ is the angle between the x axis in the local coordinate system and the X axis on the global coordinate system. This kinematic model assumes that the rate of change of the global angle θ is equal to the rate of change of the actuated angular velocity. Another assumption is that all the forces acting on the prototype is in the local x direction. This assumption is valid for lower speeds where $F_y \approx 0$.

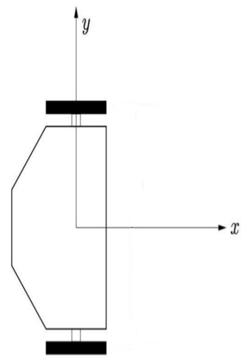


Figure 2.11: The differential drive model axis setup, modified Figure from [15].

$$\begin{aligned} \dot{x} &= u_1\, cos(\theta) \\ \dot{y} &= u_1\, sin(\theta) \\ \dot{\theta} &= u_2 \end{aligned} \tag{2.9}$$

3

The Robot Platform

To implement a control method in practice outside the simulation environment, a robot platform is required. In this chapter the robot platform used is presented, which includes the hardware, communication, and the overall system platform.

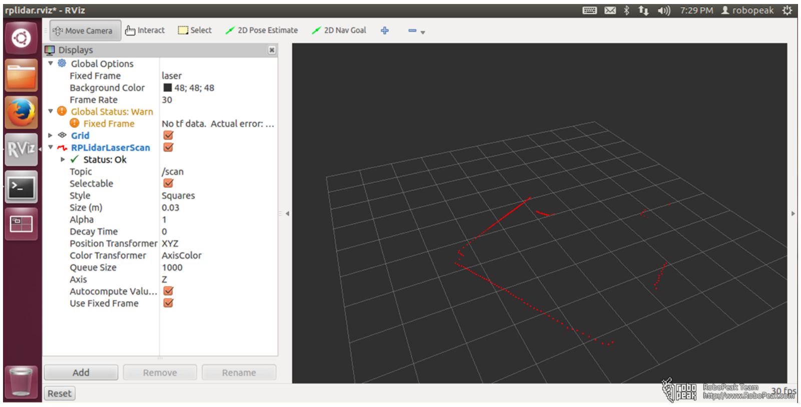
3.1 Turtlebot

The Turtlebot is a robot kit with open-source software. In this work the robot kit and one additional hardware was utilized. The robot kit includes a local computer (ASUS E202S, see Subfigure 3.2b), three-dimensional camera, electrical motors and a structural base where the all hardware is mounted on (Subfigure (3.2d) and (3.2e)). The additional hardware is the 2π rad laser scanner.

When the Turtlebot is operating the local computer is put on the top of the robot. This is because the communication from the local computer to the hardware is connected using USB connection, see Subfigure (3.2f).

The 2π rad laser scanner is a LIDAR sensor, which outputs data representing the surroundings in 2π rad. In addition, it samples $2000\,s^{-1}$ and does approximately five revolutions per second and as a result has a resolution of approximately one degree. Thus resulting in a one degree resolution, i.e., between every scanning beam from the LIDAR, there is one degree of difference, visualized in rviz [10] (Figure 3.1). The specific hardware name of the LIDAR is RPLidar version 1.

The local computer is the component which communicates with the hardware and starts up programs necessary to operate the Turtlebot. It also communicates with the remote computer, see Subfigure (3.2a). The time delay of the communication between the remote and the local computer is crucial therefore a router is used, see Subfigure (3.2c), which decreases the time delay and increases signal strength.



Global Options	
Fixed Frame	laser
Background Color	■ 48; 48; 48
Frame Rate	30
Global Status: Warn	
Fixed Frame	No tf data. Actual error: ...
Grid	✓
RPLidarLaserScan	✓
Status: Ok	
Topic	/scan
Selectable	✓
Style	Squares
Size (m)	0.03
Alpha	1
Decay Time	0
Position Transformer	XYZ
Color Transformer	AxisColor
Queue Size	1000
Axis	Z
Autocompute Valu...	✓
Use Fixed Frame	✓

Figure 3.1: The LIDAR data visualized in rviz[10]. The red dots represents all the beams which are send out by the LIDAR. In this case there is no global coordinate system. Thus, there is no visualization of a map.

The Turtlebot has two independent motors, where each motor controls one wheel. The motors are independent of each other, thus making it possible to have different speeds on the two wheels. The movement of the robot is determined by the speed of the wheels. E.g., to turn, the robot has different speeds on the wheels, and applying the same speed on the wheels results in a straight line movement. From the ROS perspective, linear speed and angular speed can be requested. The requested linear and angular speed is maintained by an internal PID controller. The maximum linear speed that can be achieved is $0.65\,ms^{-1}$ according to the data sheet [11].

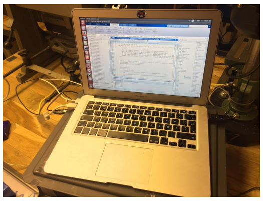
3.2 Robotic Operating System

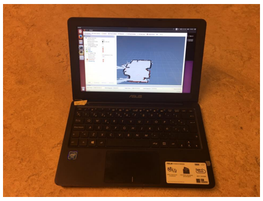
Robotics Operating System (ROS), is an open-ended collaboration framework project for writing robot software. This makes ROS popular because there is already pre-written code which can be used directly by others without knowing the details of the implementation.

For instance, a mapping algorithm from a paper in a journal might be implemented by a researcher group, while another researcher group applies it, to develop a recognition algorithm, without knowing the details of the mapping algorithm.

ROS has three levels of concepts, where two of them will be brought up here briefly because of the importance of them in Section 3.3. All the concepts discussed below are from [20].

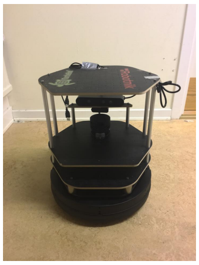
The ROS File System Level, defines a file syntax, the package, as a main


(a) Remote computer, Mac Air Mid 2012 running Ubuntu 14.04 LTS.


(b) Local computer, ASUS E202S.

(c) Router, NETGEAR, used between the remote computer and the local computer.

(d) The Turtlebot robot from the back view.


(e) The Turtlebot robot from the front view.

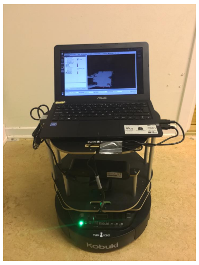

(f) The Turtlebot robot and the local computer together.

Figure 3.2: Pictures of components over the overall platform, see Figure (3.4)

unit for organising software in ROS. Such as processes, ROS-dependent libraries and anything else that is usefully organized together. E.g., the source code implementing the adaptive MCL, which would be the process that calculates the localization of the robot but also required ROS-libraries.

The communication in ROS is a peer-to-peer network of ROS processes that are processing data together and is called Computation Graph Level.

Two of the concepts in Computation Graph Level will be briefly brought up; nodes and topics. The node is a process, which processes data and publishes it but can also receive data. As for the topic, the message is routed via the topic. For instance a node publishes data to a topic and another node subscribes to the topic to receive data, see Figure (3.3), [20].

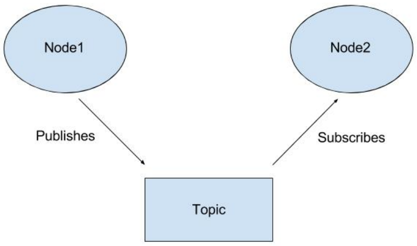


Figure 3.3: The processes (Node1 and Node2) process data and the topic (Topic) is a message container for messages between processes.

3.3 Overall platform structure

Task, Hardware and Main program communication

The overall platform structure is built by three layers; the top, middle and bottom layer (Figure 3.4). At the top layer, the remote computer operates in Matlab, where it utilises a control strategy to generate a control signal. The remote computer is connected to ROS, where ROS is running on the local computer.

As for the local computer, it runs all the processing in ROS, e.g., pose estimation and mapping.

Turning to the Turtlebot robot, it collects data from the LIDAR and wheel encoders and sends the data to the local computer for processing. The Turtlebot also receives actuator signals which it acts on.

Node and Topic communication

The communication in ROS, from a node and topic perspective, can be seen in (Figure 3.3 and 3.6). At initialization, the node `/map_server` publishes the map from the location given by the environment variable `TURTLEBOT_MAP_FILE`, set in the local computer. This map is requested by the `/amcl` node, the AMCL algorithm, in order for the `/amcl` node to work properly. See Figure (3.8) to see the map used in this work. The node `/static_transform_publisher_1488...` publishes that the transform from the Turtlebot to the LIDAR is zero, e.g., the LIDAR has the same position as the Turtlebot. The node `/amcl` is getting the LIDAR scan from the topic `/scan` which the `/rplidar` is subscribing to. The topic `/particlecloud` is where the all the posterior states are kept and maintained by the node `/amcl`. Now, the velocity is set from Matlab which subscribes to `/mobile_base/commands/velocity/`, which is illustrated by the manually added `/remote_computer`, see Figure (3.6).

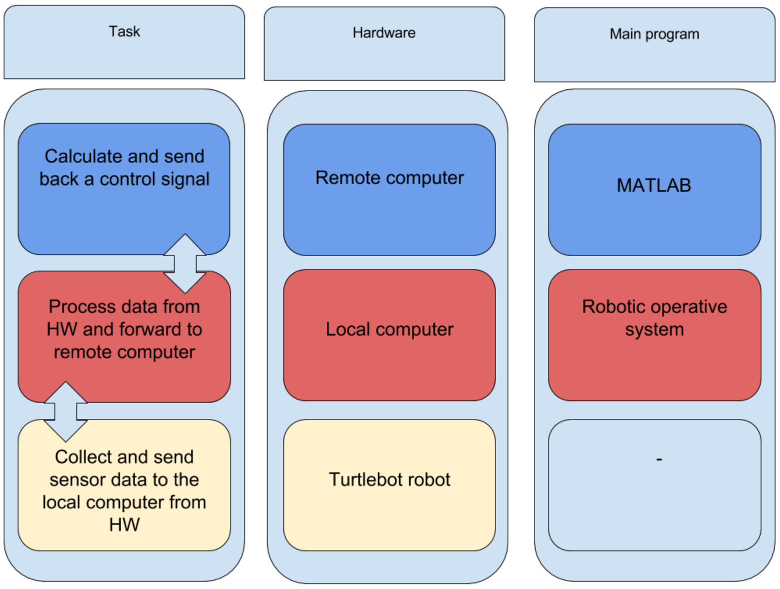

Task
Hardware
Main program
Calculate and send back a control signal
Remote computer
MATLAB
Process data from HW and forward to remote computer
Local computer
Robotic operative system
Collect and send sensor data to the local computer from HW
Turtlebot robot
-

Figure 3.4: Overall structure of the platform from a Task, Hardware and main program perspective. Notice that the main program is not commented for the Turtlebot robot since the robot operates under many programs simultaneously.

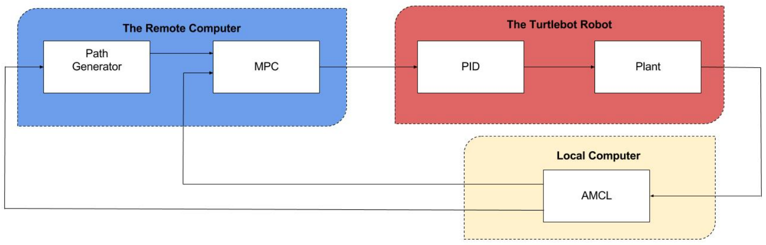


Figure 3.5: The overall structure of the platform from a block-diagram perspective.

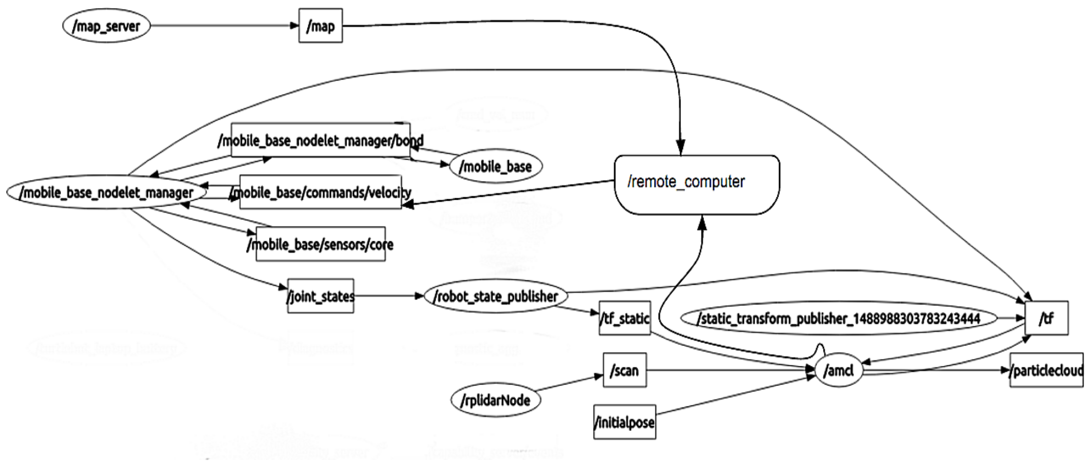

/map_server
/map
/mobile_base_nodelet_manager/bond
/mobile_base
/remote_computer
/mobile_base_nodelet_manager
/mobile_base/commands/velocity
/mobile_base/sensors/core
/joint_states
/robot_state_publisher
/tf_static
/static_transform_publisher_1488988303783243444
/tf
/scan
/amcl
/particlecloud
/rplidarNode
/initialpose

Figure 3.6: The ROS computation graph with modifications on the Figure, the node `/remote_computer` was inserted manually to give better understanding of the overall communication. The main nodes are `/mobile_base`, `/amcl`, `rplidar` and `/remote_computer`. The `/mobile_base` node operates the motors of the robot and `/amcl` generates the hypothesized poses using AMCL. The node `rplidar` outputs the measurements coming from the LIDAR. Also the node `/remote_computer` generates control signals to the robot and listens to the best pose from the `/amcl` node.

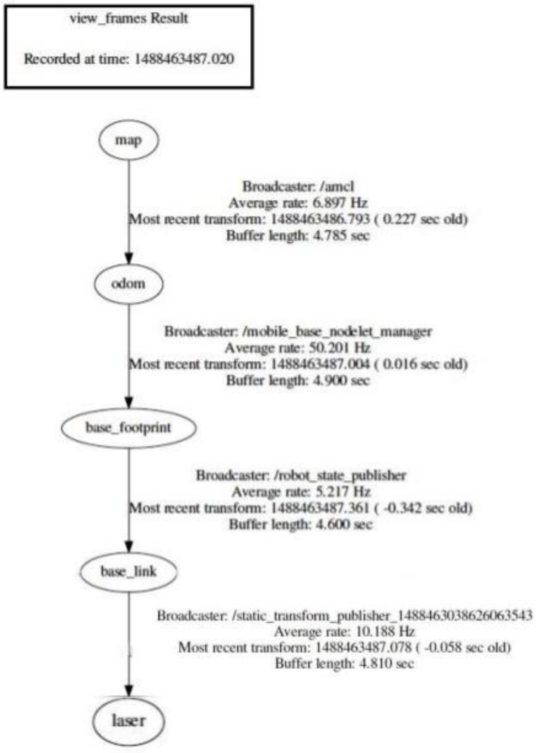

view_frames Result
Recorded at time: 1488463487.020
map
Broadcaster: /amcl
Average rate: 6.897 Hz
Most recent transform: 1488463486.793 (0.227 sec old)
Buffer length: 4.785 sec
odom
Broadcaster: /mobile_base_nodelet_manager
Average rate: 50.201 Hz
Most recent transform: 1488463487.004 (0.016 sec old)
Buffer length: 4.900 sec
base_footprint
Broadcaster: /robot_state_publisher
Average rate: 5.217 Hz
Most recent transform: 1488463487.361 (-0.342 sec old)
Buffer length: 4.600 sec
base_link
Broadcaster: /static_transform_publisher_1488463038626063543
Average rate: 10.188 Hz
Most recent transform: 1488463487.078 (-0.058 sec old)
Buffer length: 4.810 sec
laser

Figure 3.7: Modified versions of the frames in ROS during operation, where only the relevant frames are displayed. E.g., `/static_transform_publisher_1488...` publishes the transform from the `/base_link` to the attached LIDAR on the Turtlebot which corresponds to the frame `laser`. While the node `/amcl` is publishing the transform from the map to the `odom` frame.

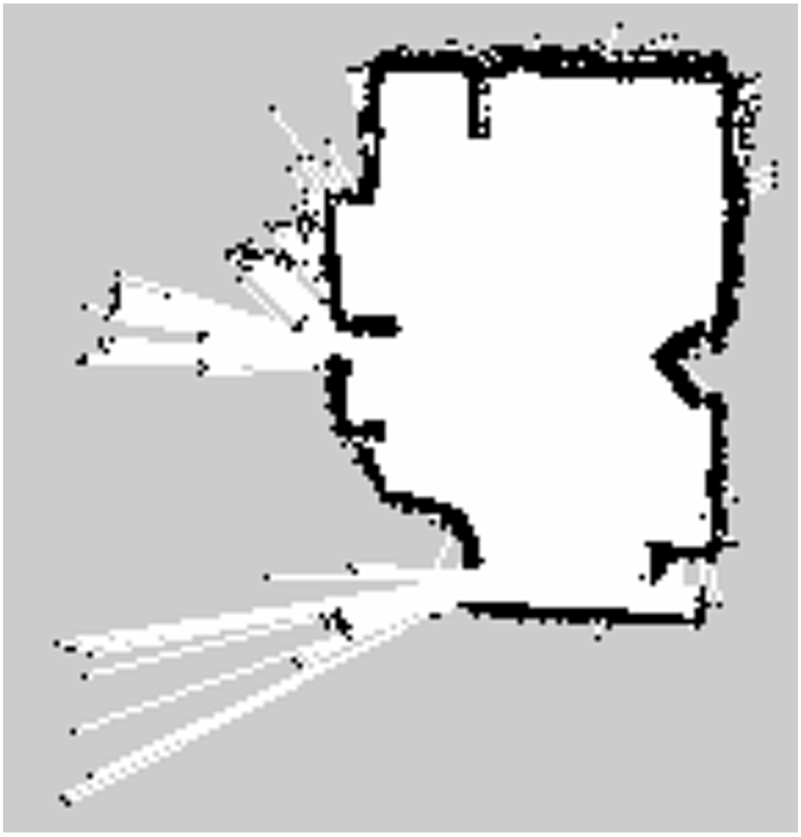


Figure 3.8: The map of the test environment during experiments.

4

Real-Time Process Model Validation

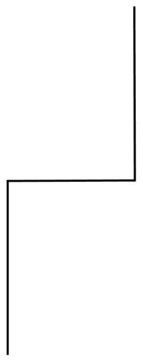
In order to have a successful Real-Time implementation the process model needs to be adequately accurate. In this chapter, the methods used to validate the process model will be outlined. A set of experiments will be outlined to validate whether the AMCL and process model give the same results in real time.

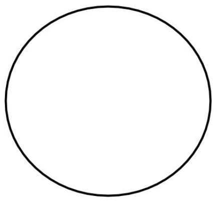
4.1 Process Model Validation

The process model is vital for the optimization and therefore the model needs to be tested for accuracy. The model to be validated is the one in Section 2.6, modeled in Simulink (Figure 4.2 and 4.3).

The measurements from the AMCL was used during the experiments, with five experiments in total, three of them are the sharp turn, circular and s-shaped motion (Subfigures 4.1a, 4.1b and 4.1c). The two last tests are from stationary spinning and straight line movement.

The control signals are predetermined for each experiment. If the process model is an accurate representation of the real process, there should be no error. I.e., the real process should act as the process model in Simulink and output the same position as Simulink does for the same control signals.


(a) The reference path in the sharp turn test.


(b) The reference path in the circle motion test.

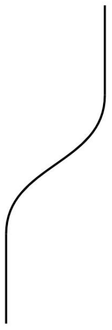

(c) The reference path in the s-shape motion test.

Figure 4.1: Notice that in the three experiments the control signals are pregenerated.

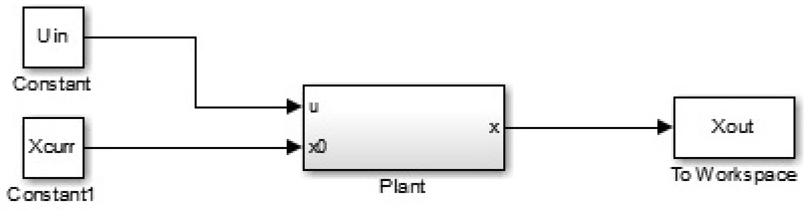


Figure 4.2: Simulink model of the differentially driven Turtlebot. For the plant structure see Figure (4.3).

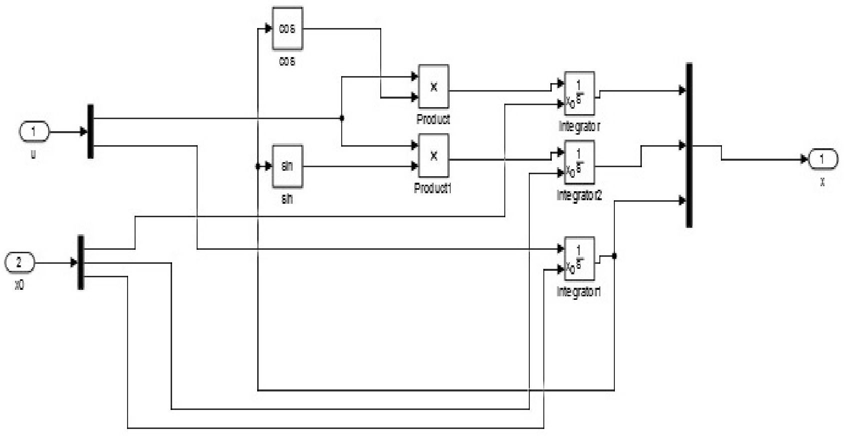


Figure 4.3: Simulink model of the differentially driven Turtlebot, the plant structure from Figure (4.2).

5

Model Predictive Controller methods

5.1 Model Predictive Control With Trajectory Tracking

Position tracking

A position tracking MPC is used to follow a trajectory defined by a reference, $r_k = [x_k^{ref}, y_k^{ref}, \theta_k^{ref}]^T$. For our purposes there will be no specified angular reference θ_k^{ref} and it will therefore be set to $\theta_k^{ref} = 0$, for all k. The resulting quadratic problem formulation

$$\begin{aligned} \text{minimize} \quad & \sum_{k=1}^{H+1} (||r_k - z_k||_Q + ||u_{k+1} - u_k||_R) \\ \text{subject to} \quad & z_{k+1} = A_k z_k + B_k u_k + c_k \\ & z_1 = z_{init} \\ & z_k \in \mathbb{Z} \\ & u_k \in \mathbb{U} \end{aligned} \tag{5.1}$$

minimizes the euclidean distance between the reference trajectory and the future state predictions, where $\mathbb{Z} = \{z : z_{min} \leqslant z \leqslant z_{max}\}$ and $\mathbb{U} = \{u : u_{min} \leqslant u \leqslant u_{max}\}$. The model used is the kinematic model in Equation (2.9). The model is approximated using Taylor series of degree 2, where

$$z_k = \begin{bmatrix} x_k \\ y_k \\ \theta_k \end{bmatrix}, \ u_k \ = \begin{bmatrix} v_k \\ \dot{\phi}_k \end{bmatrix} \tag{5.2}$$

The model is initially linearized around one point. After the first optimization the matrices are linearized around the predicted outputs of the state and decision trajectory at time k which will result in a better approximation of the model.

$$A_k = \begin{pmatrix} 1 & 0 & -v_k^o\, sin(\theta_k^o)\, h \\ 0 & 1 & v_k^o\, cos(\theta_k^o)\, h \\ 0 & 0 & 1 \end{pmatrix}, B_k = \begin{pmatrix} cos(\theta_k^o)\, h & 0 \\ sin(\theta_k^o)\, h & 0 \\ 0 & h \end{pmatrix} \tag{5.3}$$

$$c_k = \begin{pmatrix} v_k^o\, cos(\theta_k^o) \\ v_k^o\, sin(\theta_k^o) \\ \dot{\phi}_k^o \end{pmatrix} \tag{5.4}$$

The weight matrices Q and R determine the relative penalization of the tracking error and control acceleration respectively. These matrices need to be positive semi-definite in order to make the problem formulation a QP and hence convex. Since we are not interested in penalizing angle deviations the latter row of Q is a zero vector.

$$Q = \begin{pmatrix} q_x & 0 & 0 \\ 0 & q_y & 0 \\ 0 & 0 & 0 \end{pmatrix}, \quad R \quad = \begin{pmatrix} r_v & 0 \\ 0 & r_{\dot{\phi}} \end{pmatrix} \tag{5.5}$$

Car tracking

Model Predictive Control (MPC) with Car Tracking is used to follow a trajectory defined by a reference car [13]. The reference car represents a trajectory with, linear velocity, angular velocity, direction and position, of each reference point. The objective is to minimize the error between the reference car and the controlled car as well as minimizing the control effort.

The nonlinear model from the controlled and the reference car, Section 2.6, is described in compact form in Equation (5.6)

$$\begin{aligned} \dot{z} &= f(z, u) \\ \dot{z}_r &= f(z_r, u_r) \end{aligned} \tag{5.6}$$

In other words, a trajectory is generated off-line. This means that there is a control reference to follow as well as a position reference, where the controlling variables are the linear and angular velocity of the car.

By using Taylor expansion, in the reference point $(\mathbf{x}_r, \mathbf{u}_r)$, the result is,

$$\dot{x} = f(x_r, u_r) + f_{z,r}(z - z_r) + f_{u,r}(u - u_r) \tag{5.7}$$

The subtraction of 5.6 from 5.7, yields,

$$\dot{\tilde{z}} = f_{z,r}\tilde{z} + f_{u,r}\tilde{u} \tag{5.8}$$

where $\tilde{z} = z - z_r$ and $u = u - u_r$.

It follows that we get a discrete-time system model,

$$\tilde{z}_{k+1} = A_k \tilde{z}_k + B_k \tilde{u}_k \tag{5.9}$$

using forward differences, where k is the sampling time and h is the sampling period. The goal is to make $\tilde{\mathbf{x}}$ go to zero, which is the same as $\mathbf{x}$ converging to $\mathbf{x_r}$.

The objective function is

$$\sum_{j=1}^{H+1} \tilde{z}_k^T Q \tilde{z}_k + \tilde{u}_k^T R \tilde{u}_k \tag{5.10}$$

with the prediction horizon H and the weighting matrices Q and R. For a block diagram of the structure of the control strategy, see Figure (5.1), for further details see [13].

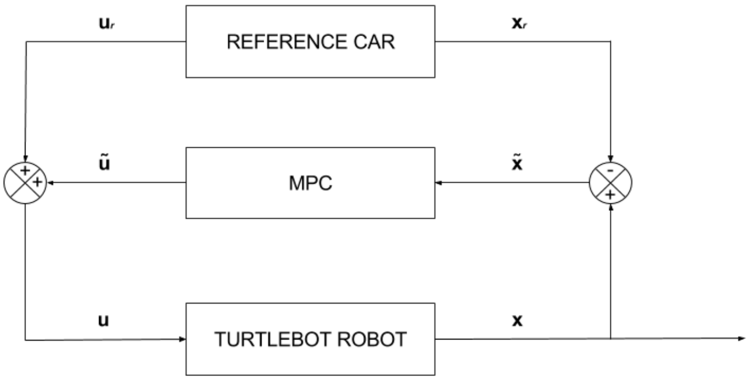


Figure 5.1: Block diagram over MPC Car Tracking, modified from [?].

5.2 Model Predictive Contouring Control

In autonomous vehicle optimization, setting a target trajectory is a challenge. There is a competing goal of tracking accuracy and traversal speed that is almost always relevant in any autonomous context. In trajectory tracking it is important that the error between the car position and the path is minimal, and for aggressive driving, such as in racing, it is also important to minimize time. Minimizing time in a MPC context is not directly possible. The optimization is always set on a specific time frame, depending on the horizon and the time step between each stage. It is usually solved by instead maximizing the distance travelled, encouraging high speeds and a greater distance between the reference points. For trajectory tracking the optimization is instead subject to closely distanced reference points to assure that the car does not deviate from the path. These seem to be contradicting problem formulations and therefore having a method to generate reference points is important. The reference trajectory can be decided by several methods, some heuristic for problems with a target speed or, as will be explored in this section, a method that does not use reference points and includes both path accuracy and maximizing speed in its objective [7], [18].

Model Predictive Contouring Control (MPCC) is a control method used for accurate tracking of a geometric path while maximizing the distance traversed. Its dual objective makes it possible for a trade off between contour accuracy and path progress. MPCC is meant to work for high speed control applications, which requires a convex problem formulation to reduce computational cost [16], [14].

MPCC has two separate control objectives, to minimize contour error and to maximize the progress along a geometric curve. The state dynamics are represented by a discrete linear system with an additional variable θ_k which denotes the relative position of the car on the geometric curve.

The following linear discrete model is used in MPCC:

$$z_{k+1} = A_k z_k + B_k u_k + c_k$$

where A_k, B_k and c_k are seen in (5.3 and 5.4)

The primary goal is to traverse along a geometric curve with minimal error. That is the first objective of MPCC, the contouring problem.

Reference path

The biaxial variables $x_c(\theta_k)$ and $y_c(\theta_k)$ that define the reference path, are determined by the path variable θ_k. The traversal variable determines the position on the function along its feasible set. The function needs to be a continuous differentiable function in order to make it possible to formulate a contouring problem around it.

The path parameter θ_k, expressed as the arc length relation for both x_c and y_c, can be determined by techniques for an interpolated function [25]. However, for the purposes of this thesis $\theta_k = x_c$.

In the simulation three continuous functions will be used to describe the desired path that the Turtlebot will follow, and in the real experiments there will be two functions. Each function will have a path parameter with $\theta_k \in [L_s, L_e]$, where L_s and L_e is the starting and end point edge of the reference curve. The angle $\phi(\theta_k)$ is the tangent line on the curve at the point $[x_c(\theta_k), y_c(\theta_k)]$, i.e.,

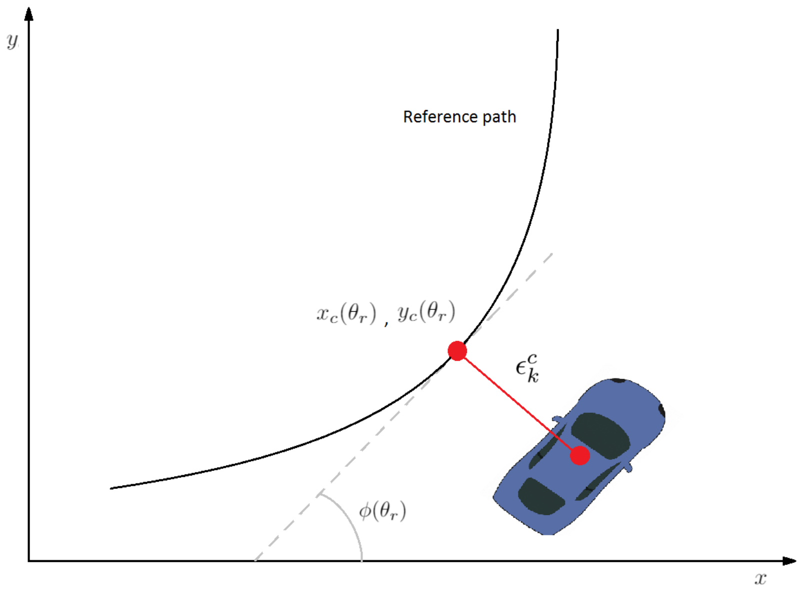


Figure 5.2: Contouring error from the car to the desired path. The contour error is the orthogonal distance between the car and the closest point on the curve.

$$\phi(\theta_k) = arctan(\frac{\Delta y_c(\theta_k)}{\Delta x_c(\theta_k)}) \tag{5.11}$$

which is used to formulate deviations from the reference function in the contouring control objective.

Contouring Control objective

The objective is to minimize the contouring error by minimizing the deviation of the car from the desired path. It is expressed as

$$\epsilon_k^c(x_k, y_k, \theta_r) = sin\phi(\theta_r)(x_k - x_c(\theta_r)) - cos\phi(\theta_r)(y_k - y_c(\theta_r)) \tag{5.12}$$

where θ_r is the path parameter that shows the point of the curve that is closest to the current position, x_k and y_k. Minimizing the contouring error will indicate that the algorithm will select control signals such that the car will converge towards the geometric function. The contouring error is depicted in Figure (5.2).

The contouring error needs to be approximated in order to be used in the optimization. The optimization will not be able to estimate the value of θ_r as it would require another embedded optimization problem. An approximation of the contouring error is:

$$\hat{\epsilon}^c_k(x_k, y_k, \theta_k) = sin\phi(\theta_k)(x_k - x_c(\theta_k)) - cos\phi(\theta_k)(y_k - y_c(\theta_k)) \tag{5.13}$$

Here is it assumed that the contouring error at the current position is approximately equal at θ_k, as seen in Figure (5.3).

Path traversing objective

To encourage the car to move along the path, a path traversing formulation is needed as well. The idea is that there is a point given by θ_k ahead of the car's current relative position on the function that the car lags behind. The error therefore corresponds to the distance to the reference point. That lag error needs to be approximated and is expressed in the following equation:

$$\hat{\epsilon}^l_k(x_k, y_k, \theta_k) = -cos\phi(\theta_k)(x_k - x_c(\theta_k)) - sin\phi(\theta_k)(y_k - y_c(\theta_k)) \tag{5.14}$$

To ensure that this approximation is sufficiently accurate we need to set weights such that $\theta_k \approx \theta_r$. We therefore want the lag error to be small, $\hat{\epsilon}^l_k(x_k, y_k, \theta_k) \approx 0$.

The Joint Problem Formulation

The dynamics in (5.2) are expanded by the following expression, i.e.,

$$\theta_{k+1} = \theta_k + v_k, v_k \in [v_{min}, v_{max}] \tag{5.15}$$

where v_k is a controllable variable and θ_k can be seen as an additional state. The value of v_k at time k is determined by the optimization. Since θ_k implicitly gives a reference coordinate at time k the variable v_k can therefore be seen as the reference control variable. The upper bound constraint of v_k needs to be set such that its maximum value is not greater than the maximum distance the robot can move in one time step. It also needs to ensure that the robot will always move forward along the path. The Turtlebot can at most move 0.2 m at each time step and therefore $|v_{lim}| = 0.2$. As seen in Figure (5.4) three functions are used to create the path for the simulations. Function 1 and 3 move in a right direction which means that $v_k \in [0, 0.2]$. Function 2 on the other hand has $v_k \in [-0.2, 0]$ to ensure that the car will move along the left of the curve. Since FORCES Pro [3] does not support setting constraint limits as parameters another controller formulation needs

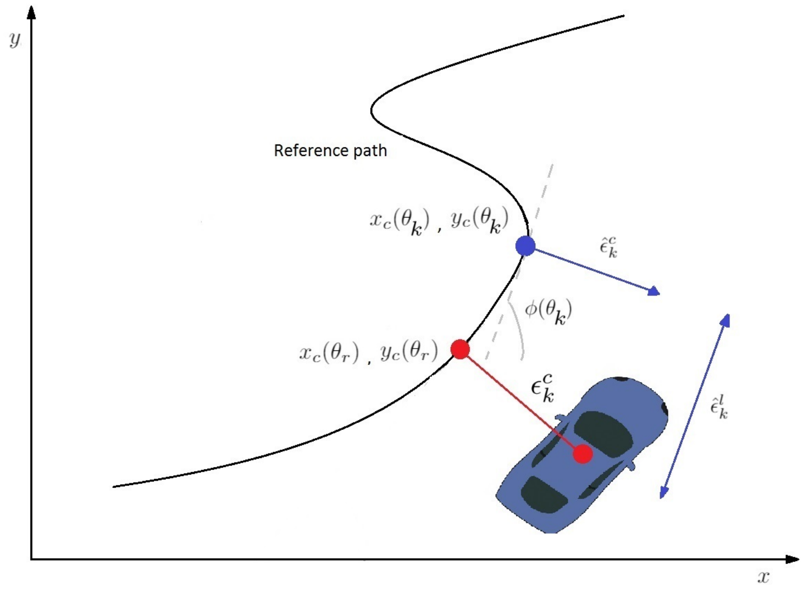


Figure 5.3: Approximated contour error and lag error. The contour error is expressed as the orthogonal distance between the closest point on the curve and the car. An approximation of the contour error is given by the reference θ_k and the reference point $[x_c(\theta_k), y_c(\theta_k)]$. The lag error is expressed as the parallel distance between the car and the reference point, with respect to the tangent angle $\phi(\theta_k)$.

to be used. The algorithm needs to switch controller on-line whenever the car starts following a new function.

The final objective includes the approximations of the contour error and the lag error. It also includes a linear term q_{θ_k}, which compliments large values of θ_k, and the weighting matrix R which penalizes large differences in u_k and v_k.

$$J = \sum_{k=1}^{N} \left(\begin{bmatrix} \hat{\epsilon}_k^c(x_k, y_k, \theta_k) \\ \hat{\epsilon}_k^l(x_k, y_k, \theta_k) \end{bmatrix}^T Q \begin{bmatrix} \hat{\epsilon}_k^c(x_k, y_k, \theta_k) \\ \hat{\epsilon}_k^l(x_k, y_k, \theta_k) \end{bmatrix} - q_\theta \theta_k + \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix}^T R \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix} \right) \tag{5.16}$$

where the weighting matrices both need to be positive semi-definite.

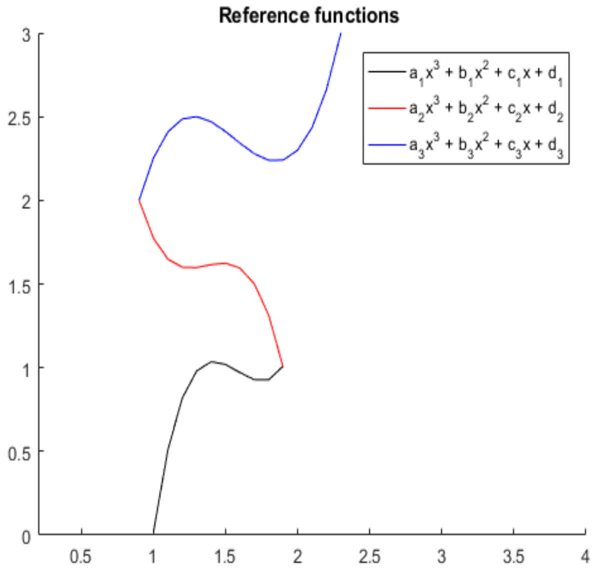


Figure 5.4: The three functions used as the reference path in the simulations.

$$Q = \begin{pmatrix} q_c & 0 \\ 0 & q_l \end{pmatrix} \quad R = \begin{pmatrix} r_{u_1} & 0 & 0 \\ 0 & r_{u_2} & 0 \\ 0 & 0 & r_v \end{pmatrix} \tag{5.17}$$

The objective can easily be expanded to include penalization of, for example, jerk. The control weighting matrix R penalizes vehicle acceleration, angular acceleration and the change of the control variable v_k. The path following weights are not as intuitive. The lag weight q_l and the tuning parameter q_{θ_k} both correspond to path speed. The parameter q_{θ_k} penalizes small values of θ_k which in turn sets the implicit reference for each iteration stage. The parameter can therefore be described as the reference weight, and will be described as such in the future. The reference weight is, however, not the primary tuning parameter regarding path speed as it only influences the reference trajectory. The lag error weight q_l is the most important parameter regarding traversal speed. As discussed before it is desirable to set q_l high to maintain a good approximation of the contour error, since when $\hat{\epsilon}^l_k(x_k, y_k, \theta_k) \approx 0$ then $\hat{\epsilon}^c_k(x_k, y_k, \theta_k) \approx \epsilon^c_k(x_k, y_k, \theta_k)$.

The resulting problem formulation is

$$
\begin{aligned}
\underset{u,v}{\text{minimize}} \quad & J \\
\text{subject to} \quad & z_{k+1} = A_k z_k + B_k u_k + c_k, \ k = 1, ..., H \\
& \theta_{k+1} = \theta_k + v_k \\
& x_k \in \mathbb{X} \\
& u_k \in \mathbb{U} \\
& v_k \in \mathbb{V}
\end{aligned}
\tag{5.18}
$$

where $\mathbb{V}$ corresponds to Equation (5.15)

A Linear Time-Variant Quadratic Program

An issue with the problem objective is that it is highly non-linear. The Hessian matrix of J_k is not positive semi-definite and the resulting problem formulation is therefore not convex. In order to reduce computational complexity an approximation of the contour error and the lag error is suggested [14]. We linearize the contour and lag error expression using a Taylor expansion of degree 2 [8]

$$
\hat{\epsilon_k}^c(x_k, y_k, \theta_k) \approx \hat{\epsilon}_k^{c'}(x_k, y_k, \theta_k) = \hat{\epsilon_k}^c(x_k^o, y_k^o, \theta_k^o) + \triangledown \hat{\epsilon_k}^c(x_k^o, y_k^o, \theta_k^o) \begin{bmatrix} x_k - x_k^o \\ y_k - y_k^o \\ \theta_k - \theta_k^o \end{bmatrix}
\tag{5.19}
$$

$$
\hat{\epsilon_k}^l(x_k, y_k, \theta_k) \approx \hat{\epsilon}_k^{l'}(x_k, y_k, \theta_k) = \hat{\epsilon_k}^l(x_k^o, y_k^o, \theta_k^o) + \triangledown \hat{\epsilon_k}^l(x_k^o, y_k^o, \theta_k^o) \begin{bmatrix} x_k - x_k^o \\ y_k - y_k^o \\ \theta_k - \theta_k^o \end{bmatrix}
\tag{5.20}
$$

The approximation can be linearized around either a point or a trajectory. Linearizing around a trajectory is preferable as it is more accurate, which is important since the contour error and the lag error expressions are approximations even in the non linear formulation. The challenge is to find the right trajectory to linearize around. This is not a trivial task since, opposed to model trajectories in the previous section, there is no real indication of the future state values in contouring control. The problem is solved by using a iterative method in which the optimization is solved around an approximated set of linear error expressions. The output will then converge towards accurate error approximations. The new quadratic problem formulation is

$$
\begin{aligned}
\underset{u,v}{\text{minimize}} \quad & J^{'} \\
\text{subject to} \quad & z_{k+1} = A_k z_k + B_k u_k + c_k, \; k = 1, ..., H \\
& \theta_{k+1} = \theta_k + v_k \\
& x_k \in \mathbb{X} \\
& u_k \in \mathbb{U} \\
& v_k \in \mathbb{V}
\end{aligned} \tag{5.21}
$$

where

$$
J^{'} = \sum_{k=1}^{N} \left(\begin{bmatrix} \hat{\epsilon}_k^{c'}(x_k, y_k, \theta_k) \\ \hat{\epsilon}_k^{l'}(x_k, y_k, \theta_k) \end{bmatrix}^T Q \begin{bmatrix} \hat{\epsilon}_k^{c'}(x_k, y_k, \theta_k) \\ \hat{\epsilon}_k^{l'}(x_k, y_k, \theta_k) \end{bmatrix} - q_\theta \theta_k + \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix}^T R \begin{bmatrix} \Delta u_k \\ \Delta v_k \end{bmatrix} \right) \tag{5.22}
$$

Algorithm 2 Implementing the Linear Model Predictive Controller

```
1: Initialize $\boldsymbol{u_0} = 0$ and $\boldsymbol{v_0} = 0$
2: Compute the corresponding state values $z_2^o ... z_H^o$ from (5.2) and $\theta_2^o ... \theta_H^o$ from (5.15)
3: Compute (5.19) and (5.20) for $k = 1...H$.
4: Solve the optimization problem (5.21) using the error estimations, receiving the output predictions $z_2 ... z_H$ and $\theta_2 ... \theta_H$
5: If $(||z_2 - z_2^o|| > \epsilon_z)$ and $(||\theta_2 - \theta_2^o|| > \epsilon_\theta)$ set $\boldsymbol{z_k^o} = \boldsymbol{z_k}$, $\boldsymbol{\theta_k^o} = \boldsymbol{\theta_k}$ and move to step 3, otherwise send control signal output $u_1$ to plant before moving to step 3.
```

For each iteration the approximation will improve until it sufficiently describes the error along the planned trajectory. The error terms ϵ_z and ϵ_θ is chosen such that adequate convergence is guaranteed and without causing too much computational delay. For our experiments we use $\epsilon_z = 0.1$ and $\epsilon_\theta = 0.1$. A visualization of the converging error approximation is shown in Figure (5.5). In the example seen in Figure (5.5) it takes three iterations in the beginning of the simulation to approximate the contour error. After that the predictions estimate the error terms well enough for future optimizations.

The MPCC problem formulation will be computed with the solver FORCES Pro [3]. The resulting problem formulation is convex and can therefore be solved by FORCES quickly. Because of its convexity it will have one global minimum which results in predictable high speeds when using efficient solvers. FORCES Pro also supports non-linear optimization with their non-linear Barrier Interior Point method, which gives the option to test out the algorithm on a more exact approximation of the contouring and lag error

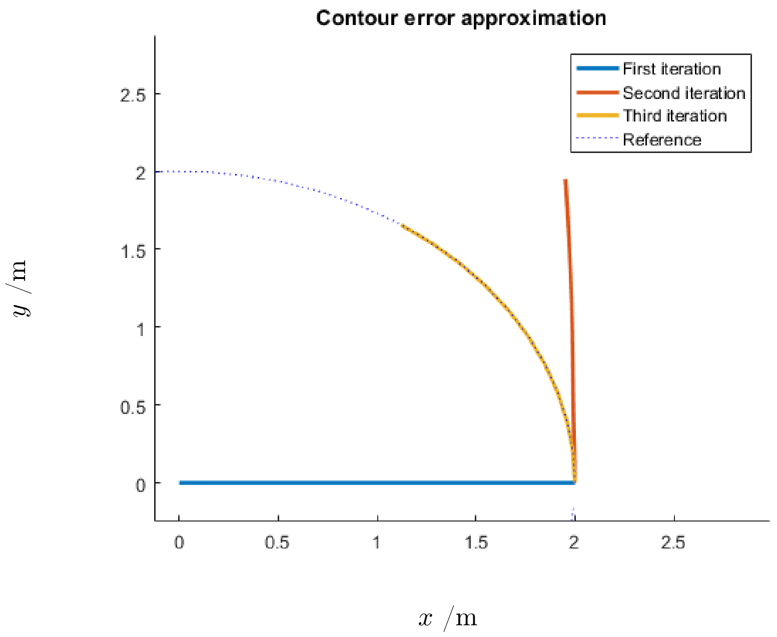


Figure 5.5: The contour error approximations of a reference circle. The first iteration results from initial control approximations of $u_k^o = 0$ and $\theta_k^o = 0$ for all k. Notice that the error approximation is large in the first iteration, since the optimization output predicts movement to $[x, y] = 0$.

(Equation (5.18)). The optimization time might, however, be unpredictable and very long.

6 Results

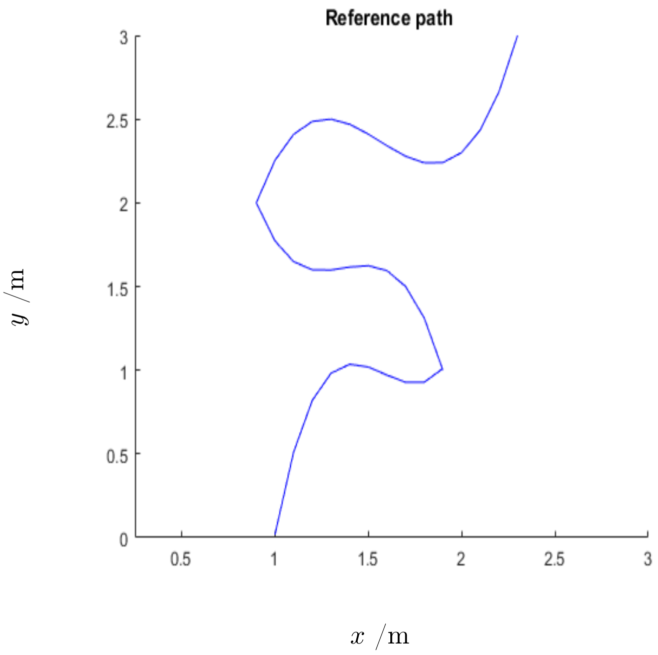


Figure 6.1: The reference path for the simulations

This chapter will present the results from the simulations and the real experiments. There are results from both the trajectory tracking and the contour tracking methods. Figure (6.1) shows the path used in the simulation. For trajectory tracking a set of reference points were taken from the path, and in contouring control three separate functions make up the path, as seen in Figure (5.4). The functions were purposely designed to have significantly different derivatives at the edge points where the functions meet, intended to test the limitations of the MPCC method. In all simulations the blue lines correspond to the reference, and the red dots are the actual measured or simulated car position.

6.1 Simulation results

Trajectory tracking

Figure (6.2) shows the trajectory tracking simulation. The reference points are generated with an Euclidean distance of $||r_i - r_{i-1}||_2 = vh$ where $h = 0.3$ is the time step and v is the target speed. The target speed is set to $v = 0.5$, and $v = 0.3$ at the sharp turns where the functions change from one to the other.

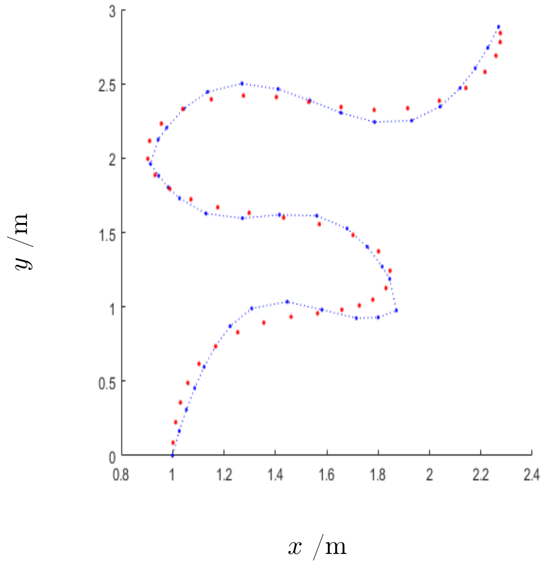


Figure 6.2: The simulation of the trajectory tracking method

Figures (6.3) and (6.4) show the contouring error and the speed respectively at a certain simulated time. In the simulation it is assumed that new control signals is set exactly at each time step. The contour error variance is quite large compared to the results shown in Figure (6.8), for the contour-

Travel time /s	Max contour error /cm	Mean contour error /cm
12.3	10.4	3.9

Table 6.1: Contour error and traversal time from the trajectory tracking simulation in Figure (6.2).

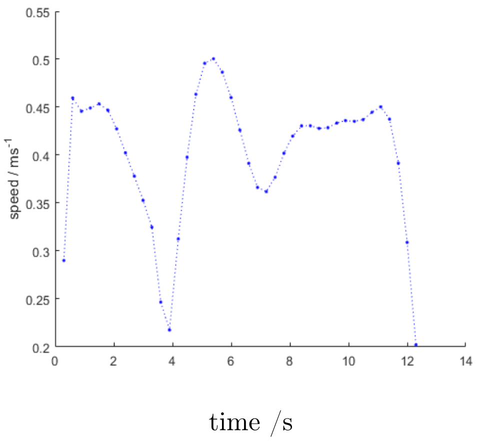


Figure 6.3: The simulated speed at each time step. Notice the the dips in the curve at $t \approx 4$ and $t \approx 7$ which corresponds to the sharp turns.

ing control simulation. Noticeable in Figure (6.3) is that the speed seldom reaches $v = 0.5$.

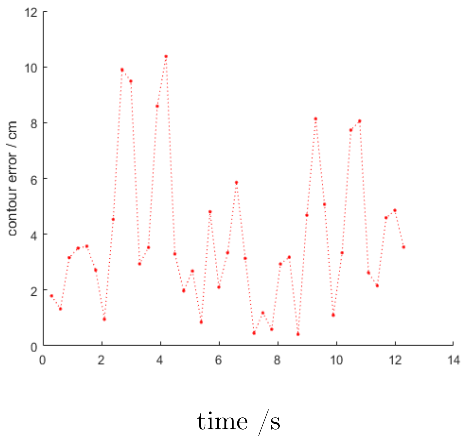
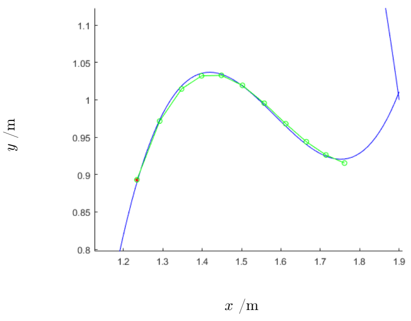


Figure 6.4: The contour error from the simulation in Figure (6.2). The two peaks at $t \approx 3$ and $t \approx 4$ indicate the error during the first sharp turn.

Contour tracking

In this simulation the non-linear MPCC problem formulation was simulated along the same reference path. Instead of reference points the algorithm solves the optimization problem around the three reference functions.



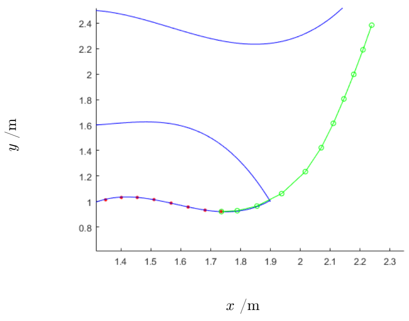


Figure 6.5: MPCC predictions of horizon $H = 10$

Figure 6.6: MPCC predictions near the function edge

Predictions for the contouring control simulation are depicted in Figures

(6.5) and (6.6). The simulation was made with a large weight on the contouring error, where $q_c = 100$, $q_l = 10$ and $q_\theta = 0.1$. The complete simulation can be seen in Figure (6.7). Figure (6.6) shows the predictions at the edge of the first function, before the reference function in the optimization has changed. One can see that it predicts further movement along the same function until it reaches the edge where $\theta = 1.9$ and the other function is the active reference.

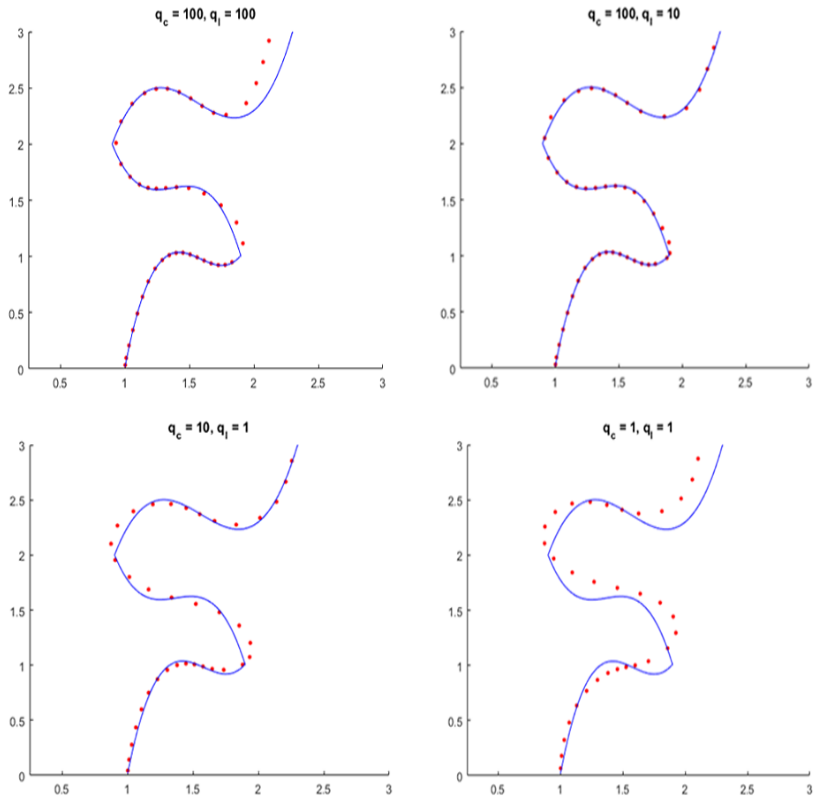


Figure 6.7: Four separate simulations of the non-linear MPCC. Notice the tracking error for the low contour weight $q_c = 1$. All subfigures have the vertical axis y and the horizontal axis x with unit m.

In Figure (6.7) there are four separate simulations depicted. For every simulation the tuning parameter $q_\theta = 0.1$. The contouring weight q_c is seen to directly affect the accuracy of the traversed path. Data from each separate simulation are seen in Table (6.2). The largest relative contour error when q_c $= 100$ and $q_l = 10$ leads to the most accurate path following, with the trade off that it takes the longest time to finish the track. Reducing the contour weight q_c affects the path speed the most, although with increased tracking error. According to the simulation the MPCC handles the reference change more effectively with a higher contour weight, which is reasonable since the speed is also decreased.

q_c	q_l	Travel time/s	Max contour error/cm	Mean contour error/cm
100	100	14.1	16.7	2.36
100	10	14.7	2.88	0.89
10	1	11.4	9.3	2.69
1	1	10.2	17.7	7.83

Table 6.2: Contour error and traversal time

q_c	q_l	Max solve time/ms	Median solve time/ms
100	100	26.0	5.6
100	10	24.6	4.3
10	1	17.0	3.2
1	1	10.8	3.7

Table 6.3: Solver computational time

For each simulation the solve time was calculated as seen in Table 6.3. The solve time was fast for each iteration giving an expected time of around 4 ms which is fast for such a non-convex problem. For the trajectory control algorithm the expected time was around 2 ms. Since both algorithms used the high level interface in FORCES pro the aptitude to solve non-linear problems is likely more efficient.

For faster traversal the reference parameter q_θ was increased to $q_\theta = 0.2$. Since $q_\theta \theta_k$ is a linear term and negatively proportional to the objective function (Figure 5.16), increasing the reference parameter q_θ would encourage larger values of θ_k. The variable θ_k can be seen as the reference points decided by the algorithm.

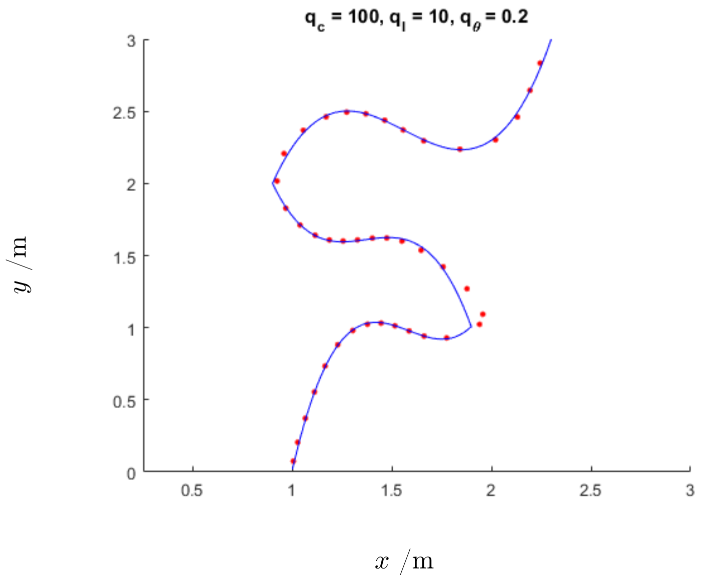


Figure 6.8: In this simulation the reference weight was increased to $q_\theta = 0.2$. The only notable path error can be seen at the meeting point of the first and second reference function.

Figure (6.8) is another simulation in an attempt to decrease traversal time while maintaining a highly accurate path following. From Table 6.4 it can be seen that the mean contouring error $\bar{e}_c$ is similar to when $q_\theta = 0.1$ with the additional effect that the traversal speed is decreased by 2 seconds.

Table 6.4: Travel time, contour error and solve time

Travel time /s	Max contour error /cm	Mean contour error /cm
12.6	7.89	1.20

Max solve time /ms	Median solve time /ms
26.0	5.6

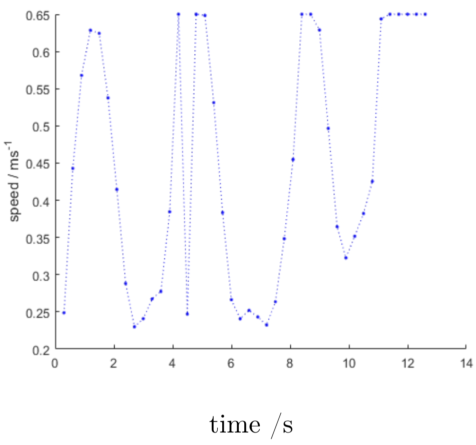


Figure 6.9: The simulated speed for the simulation in Figure 6.8.

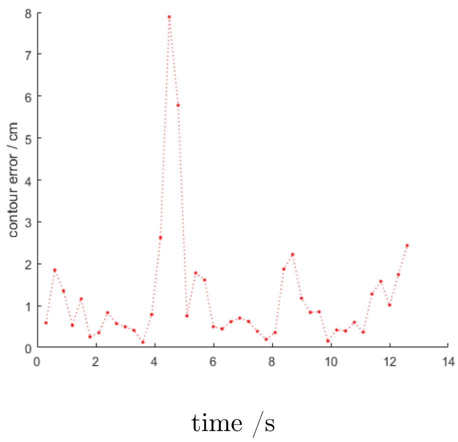


Figure 6.10: The contour error for the simulation in Figure (6.8).

As seen in Figure (6.9) the speed varies relatively smoothly during changes in function curvature, except at the point where the reference functions changes. That is also the point with the only significant contouring error of 7.89 cm (Figure 6.10).

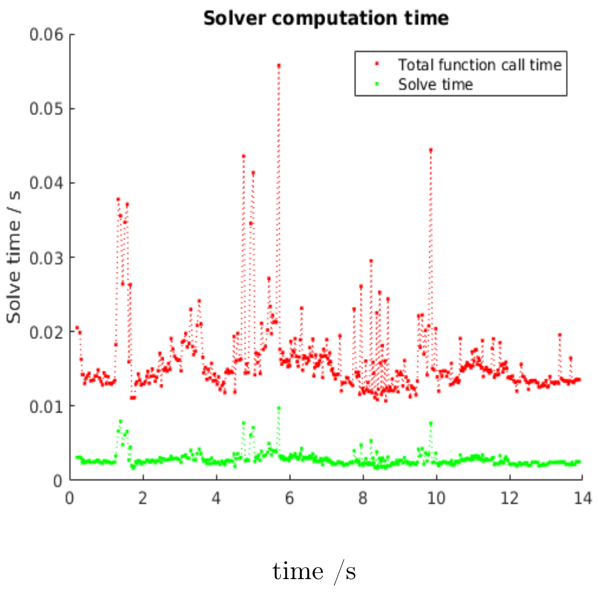


Figure 6.11: The computational time for the LMPCC controller. Notice the difference in the solve time vs the function call time.

6.2 Experiment results

Contouring control

In the real time experiments we used another path than for the simulations, created by two cubic polynomials. As can be seen in Figure (6.12) the vehicle starting position is $[x_{start}, y_{start}] \approx [1.3, 3.5]$ and the reference function changes at $\theta_k = x_c = 2.56$. For the five experimental tests different weights were chosen to tune the controller for different control circumstances.

The non-linear problem formulation was not suitable for real-time experiments. It has a fast solve time from FORCES Pro but there was a significant delay for calling the solver from Matlab. The function call was estimated in a range from 50-200 ms which made it very difficult to successfully implement the control strategy on-line. In the experimental tests we have therefore solely used the linear approximation. In the LMPCC implementation the function call time was significantly faster, although it also gives a notable delay to an otherwise fast solver. Figure (6.11) gives an idea of the typical function call time vs solve time in LMPCC. Maximum function call time is around 60 ms.

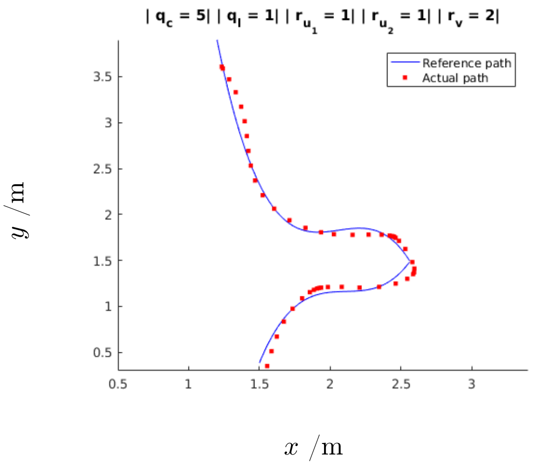


Figure 6.12: The first LMPCC with a path tracking objective. The vehicle moves in a downward direction

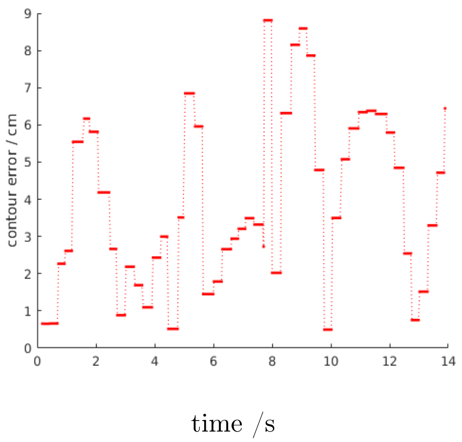


Figure 6.13: The contouring error in Test 1

Test 1 In Figure (6.12) relatively low contour and lag weights were used , $q_c = 5$, $q_l = 1$. A larger penalty was put on the angular velocity r_{u_2} penalizing sharp turns which can be seen in the figure. The intial pose of the Turtlebot comes with a slight angular deviation from the tangent of the function. Figure (6.13) shows the contouring error over time traveled.

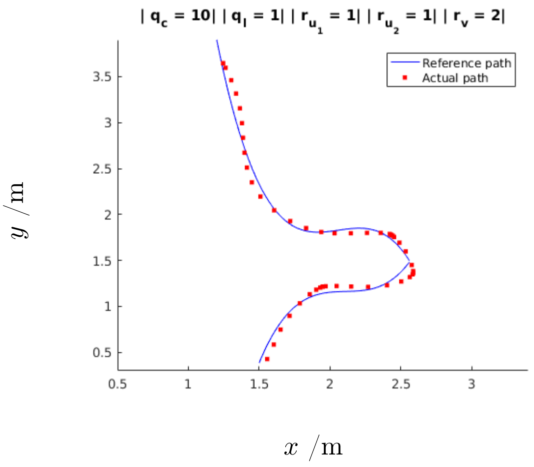


Figure 6.14: Path trajectory in the second test with a doubled contouring weight, $q_c = 10$

Test 2 In Test 2 we increased the contouring weight to $q_c = 10$ which gave in general a more accurate reference tracking. The only notable error is when the reference function change at $x = 2.56$.

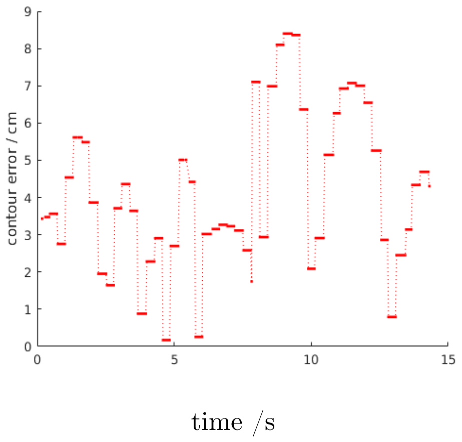


Figure 6.15: Contouring error in Test 2. A larger contouring weight is shown to increase the path accuracy.

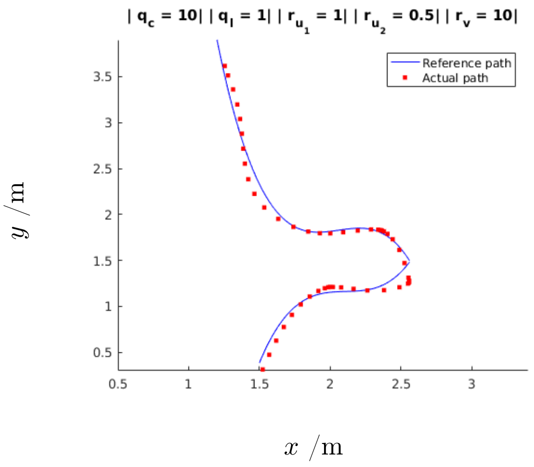


Figure 6.16: Path trajectory in Test 3 with a decreased $r_{u_2} = 0.5$.

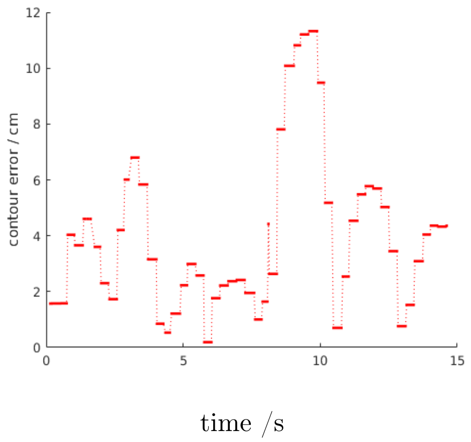


Figure 6.17: Contouring error in Test 3. The error is clearly greater than in Test 2.

Test 3 Test 3 is similar to test 2 except for the decreased penalty of the angular velocity. The increase in tracking error suggest some model or position estimation inaccuracies during greater angular velocities.

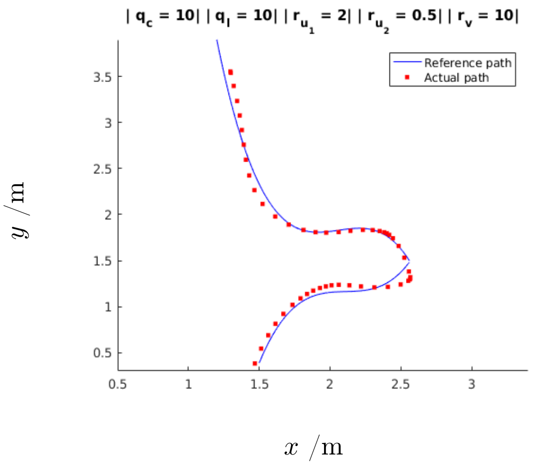


Figure 6.18: Path trajectory in Test 4 with increased lag error weight $q_l = 10$.

Test 4 In Test 4 the lag weight q_l was increased to $q_l = 10$ giving the result of a slightly better tracking than in Test 3. This is likely due to a better approximation of the contouring error which we have for larger lag error weights. Having a sufficiently large contouring weight also gives the result of accurate tracking.

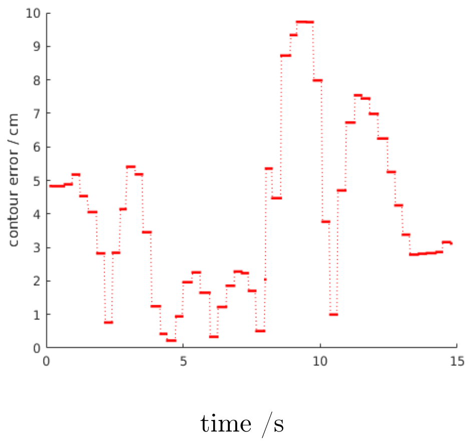


Figure 6.19: The contouring error in Test 4. The error is largest at the sharp turn where the reference functions change.

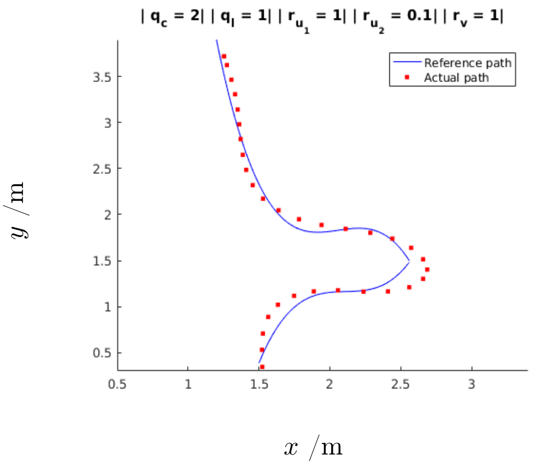


Figure 6.20: Path trajectory in Test 5 with a smaller contour and lag error weight, leading to faster travel and larger error.

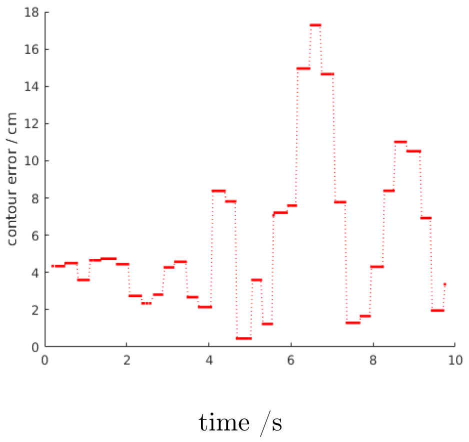


Figure 6.21: Contour error in test 5. The largest error from all the tests

Test 5 In Test 5 a smaller contouring weight was set, $q_c = 2$, giving a rapid traversal of the course with the expense of a large contouring error. The error is still at most $e_c = 1.7\ dm$ and could likely be used for an autonomous racing application.

	q_c	Travel time/s	Max solve time/ms	Median solve time/ms
Test 1	5	13.9	9.7	2.6
Test 2	10	14.3	18.2	2.5
Test 3	10	14.6	14.2	2.6
Test 4	10	14.8	13.0	2.6
Test 5	2	9.8	6.0	2.9

Table 6.5: The measured travel time and solving time for each LMPCC test

Tables (6.5) and (6.6) present data from each test. The results are generally as expected as the travel time increases for larger contouring weights q_c, with the additional effect of more accurate tracking. The solve time is in general around 2-3 ms for each test run. Comparing to the simulation of the non-linear MPCC it is about half the solve time.

	q_c	Max e_c/cm	Mean e_c/cm	Mean speed/ms^{-1}
Test 1	5	8.81	3.99	0.365
Test 2	10	8.40	4.28	0.378
Test 3	10	11.30	3.86	0.354
Test 4	10	9.73	4.04	0.33
Test 5	2	17.28	5.72	0.606

Table 6.6: The estimated contour error and average speed for each LMPCC test

Trajectory tracking

In this test the trajectory tracking control method was tested using FORCES Pro. The same path was used as in the LMPCC with the difference being that reference points were collected from the two reference functions. Again as in the simulation tests the reference points were generated with a Euclidean distance $||r_i - r_{i-1}||_2 = vh$ where $h = 0.3$ is the time step and v is the target speed. The set target speed was $v = 0.5$, and $v = 0.3$ close to where the reference function change.

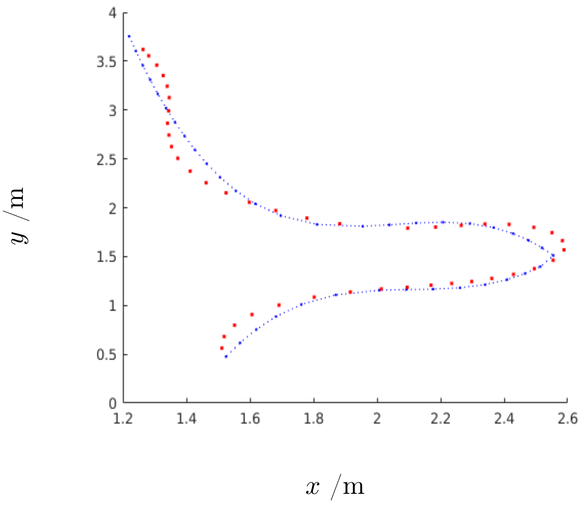


Figure 6.22: Path trajectory from the trajectory tracking controller

Travel time/s	13.5
Max solve time/ms	12.7
Median solve time/ms	4.0
Max e_c/cm	10.5
Mean e_c/cm	4.14
Mean speed/ms^{-1}	0.389

Table 6.7: Solve time, contour error, travel time and average speed for the trajectory tracking control implementation

6.3 Real-Time Process Model Validation Results

Five experiments were made to determine the model validity. Since the measured position is just an estimate we used different noise variance parameters in the first three experiments to also determine the best position estimation. The speed was calculated from the position estimates and compared with the set control speed. The comparison gave an indication of the accuracy of the kinematic model.

	$\alpha_1 - \alpha_4$
Test1	0.2
Test2	0.1
Test3	0.05
Test4	0.3

Table 6.8: The different sets of α values for the first three experiments, see Subfigures **??**, 6.23b and 6.24b.

Sharp turn

The model accuracy was tested for very sharp turns. In Subfigure (6.23b) the general characteristics is displayed for all the different sets of parameters $\alpha_1 - \alpha_5$.

The error between the process model versus the AMCL velocity estimation is large during large speed deviations, Subfigure (6.23a). During this experiment, Test 4 with $\alpha_1 - \alpha_4 = 0.3$ obtained the best result under sampling of 0.3 s. That is despite that the parameter set assumes the initially largest variance scaling in the noise.

Circular motion

Tests were made with constant linear and angular speed. In Subfigure (6.23a) it is evident that there were position deviations starting at around 8 s of the testing period. At this time the estimated velocity is twice as fast as the simulated velocity.

Looking at the Subfigure (6.23b) the smallest noise variance, Test 3, is the one that most accurately follows the model of the process, for both the period $h = 0.2$ and $h = 0.3$ s.

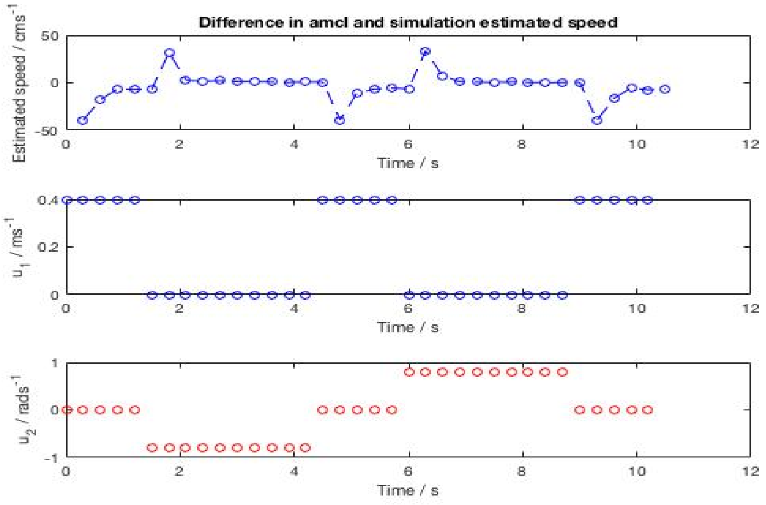
S-shape motion

In the s-shape experiment the velocity estimation from the AMCL is tested with a constant speed and varied angular velocity.

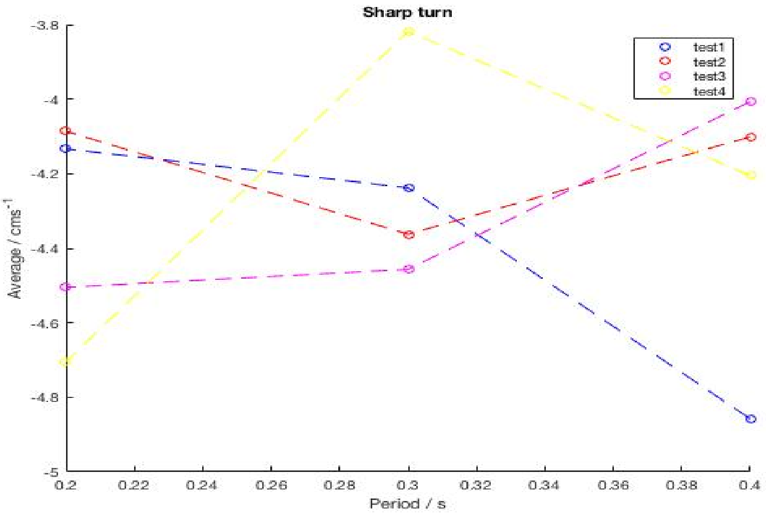
As seen in Subfigure (6.24a) the velocity error converges towards zero. The parameter set in test 2 produces the smallest error approximately for all the sampling times (Figure 6.24b).

Stationary spinning motion

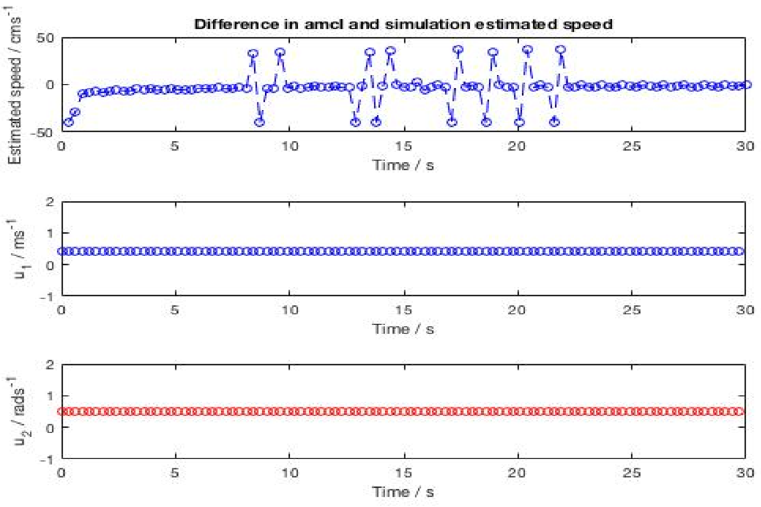
In this experiment the position estimation from the AMCL and the process model is measured when the Turtlebot is spinning with zero linear velocity, Figure (6.25). The period time $h = 0.3$ s and the parameter set in Test 3 is used, Table 6.8. The position is shown to deviate significantly for large angular velocities.



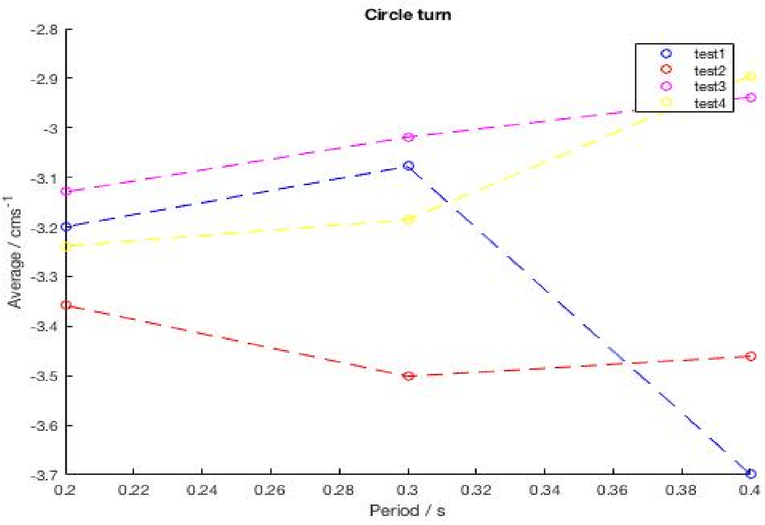
(a) Top: The difference in estimated speed between the AMCL package and the process model during the sharp turn experiment.
Middle: The control signal u_1 (linear speed) during the sharp turn experiment.
Bottom: The control signal u_2 (angular speed) during the sharp turn experiment.



(b) The average difference in speed from the AMCL package and the process model with different α_1 - α_4 and sampling rate.

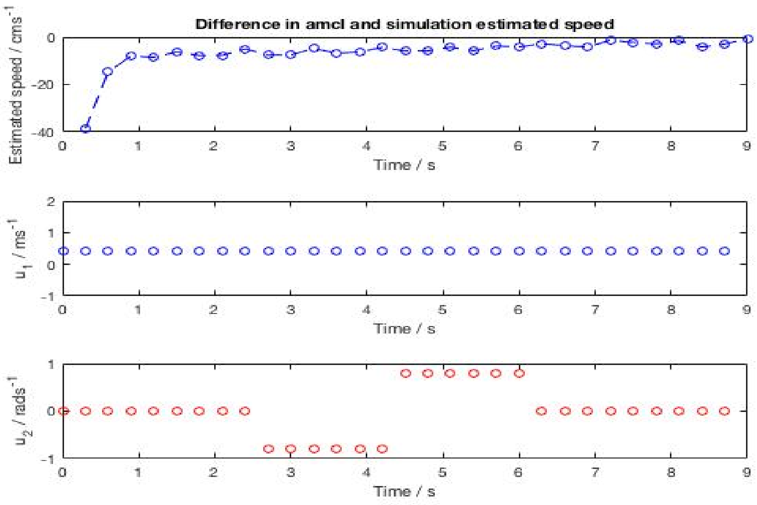


(a) Top: The difference in estimated speed between the AMCL package and the process model during the circle motion experiment.
Middle: The control signal u_1 (linear speed) during the circle motion experiment.
Bottom: The control signal u_2 (angular speed) during the circle motion experiment.

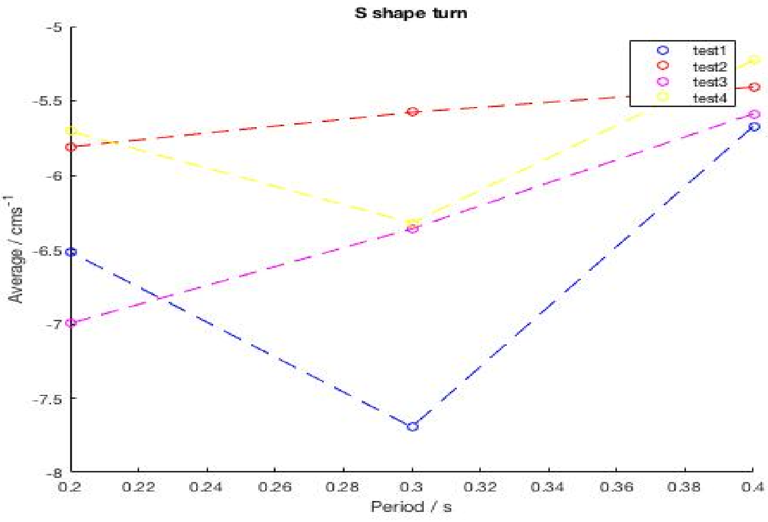


(b) The average difference in speed from the AMCL package and the process model with different $\alpha_1 - \alpha_4$ and h.

Figure 6.23: The results from the circle motion experiments.

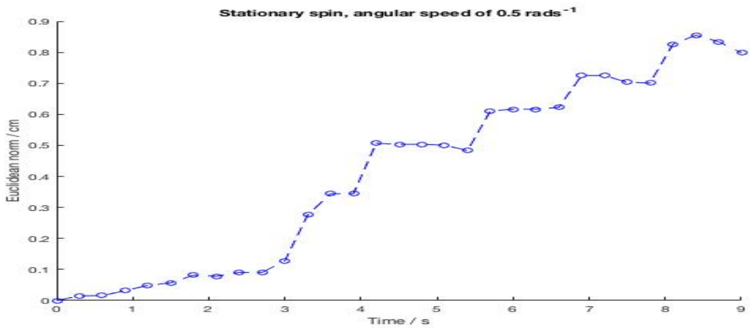


(a) Top: The difference in estimated speed between the AMCL package and the process model during the circle motion experiment.
Middle: The control signal u_1 (linear speed) during the circle motion experiment.
Bottom: The control signal u_2 (angular speed) during the s-shape motion experiment.

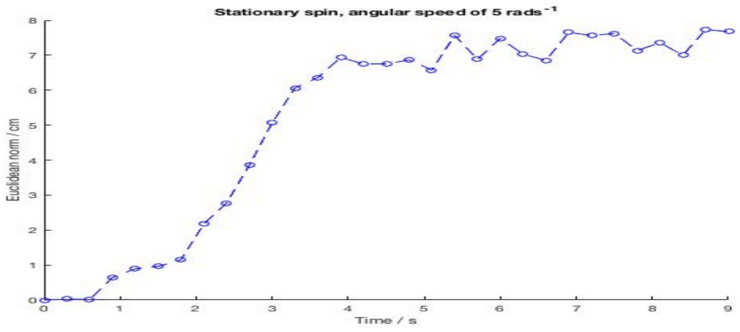


(b) The average difference in speed from the AMCL package and the process model with different $\alpha_1 - \alpha_4$ and h.

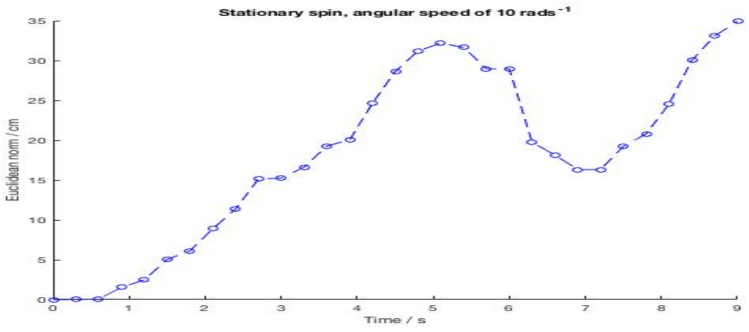
Figure 6.24: The results from the s-shape motion experiments.



(a) Notice that the AMCL estimates that the robot has moved approximately 0.9 cm when in reality it is staionary. The robot rotates with half a radian per second.



(b) For a higher rotational velocity the position error rapidly increases and converges at around 7 *cm*.



(c) For a rotation of 10 radians per second the error becomes very large.

Figure 6.25: The Euclidean norm between the AMCL and process model position estimation for different angular speeds.

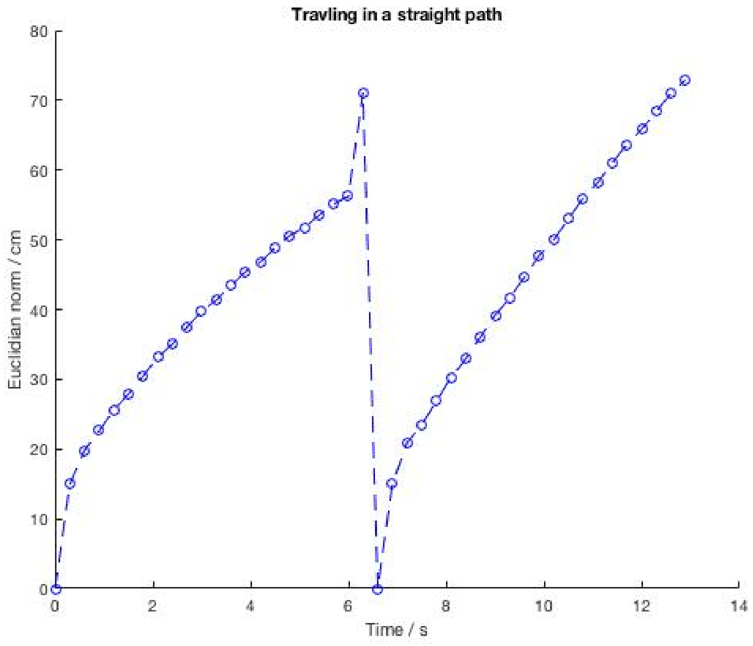


Figure 6.26: The Euclidian norm between the process model in Simulink and the AMCL position estimation when the robot is driving straight, turns π radians, and then travels straight again. When the robot turns, the position is not sampled, therefore, the error is again zero in the middle of the experiment.

Straight line movement

Figure (6.26) shows the euclidean norm between the position estimator, AMCL and the process model. The Turtlebot is seen to be drifting while driving straight forward as can be seen in Figure (6.26).

7 Discussion

7.1 Simulation results

The most noticeable result in the simulation is that the non-linear problem formulation of the MPCC problem was able to compute each optimization at a speed that would be sufficient on-line. This is surprising since the problem formulation in Equation (5.18) is not convex, but highly non-linear. The contouring error expression is subject to the geometric function that is of third order, and the term $sin(\phi(\theta))$ involves the sine of a second order polynomial. The lag error formulation is equally non-linear. The resulting computation time is still not more than a few milliseconds, as seen in Table (6.3). On average the speed of the computations is around 2-3 ms slower than the other optimization problems, with QP formulations.

In the simulation seen in Figure (6.8) the median computational time is 5.6 ms while the maximum computation time is 26.0 ms. It is noticeable that the solve time has a much higher variance for MPCC than in the trajectory tracking QP problem, which had a maximum solve time of 4.6 ms and median 2.4 ms (Figure 6.2). 26.0 ms is still a relatively low computational time and sufficient for real-time control with the time step h used in the experiment, 300 ms. Looking at the output data the solver takes around 40-100 iterations to solve the MPCC problem with the default built it tolerance of the solver.

Explaining the computational speed of the non-linear MPCC problem requires further analysis. The Hessian was calculated and it was not positive semi-definite, meaning that it is not convex. FORCES Pro mainly uses the solver method Barrier Interior Point Method (IPM). In the high lever interface of FORCES Pro, in which MPCC was implemented, a non-linear Interior Point method is the main solver. The non-linear solver is designed to solve non-convex optimization problems as well. The speed at which the problem is solved is therefore most likely due to the efficient solver as the non-linear barrier IPM is designed for applications in real time. Other causes for the rapid solving time is choice of using the differential drive model, while using a bicycle model with added dynamics for longitudinal and horizontal slip might make the problem formulation more complex.

Issues with non-linear optimization

Even if the solve time is very small there are other issues regarding general non-linear optimization. You can never guarantee that it is reliable in a real time environment. However, the speed and efficiency of a quadratic problem is the reason why these problem formulations are widely used. They are guaranteed to be convex, and guaranteed to have a minimum. A non-convex optimization problem does not necessarily have a minimum, and solving one does not guarantee that you will ever find a solution, let alone the global one. That brings up a secondary issue; there might be several minima and the one the solver finds might not be the global minimum.

Furthermore, as the controller is implemented in Matlab there is a delay from the function calling the FORCES Pro solver, which prevented us from using the non-linear MPCC formulation in real time. The function call time is about 10 times longer than the actual solving time and was therefore not used in the real time experiments. These issues do not, however, exist when the solver is implemented on an embedded system. In an embedded system the solver is called directly from the algorithm which removes the delay. The function call time is remarkably lower for QP formulations, which was around 30 ms. A smaller time between measuring the position and calculating the control signals is preferable. Problems arose when the function call time reached over 100 ms, which it often did during computing the non convex problem.

Reference functions

The errors in the simulation were always the largest during the change from one reference function to another. This is expected since the optimization only takes into account the current function curvature. Choosing reference functions with smoother curvature transitions would improve the results. Looking at Figure (6.6) we can see that the predictions follow the function curvature after the edge where it will not longer be a reference. Since the derivative of the edge point between the two functions are significantly different the contouring control method will, at a few time steps, have a larger error.

7.2 Experimental results

For the real-time experiments the LMPCC was tested with decent results, althouth the function call time is still an issue, as can be seen in Figure (6.11). The function call time comes from the time which Matlab spends inside the generated MEX function. Since the method iterates through perhaps several error approximations until it converges this delay might accumulate and become a problem, although in the experimental results this was not noticeable. It would still be preferable to implement the controller method on the embedded system in C++ to reduce computational time.

The result were mostly as expected as the tracking accuracy was improved with larger values of the contouring weight q_c. For $q_c = 2$ and $q_l = 1$ (Figure 6.20) the travel time was the lowest with 9.76 s, with an average speed of 0.606 ms^{-1}. These tuning parameters were therefore suitable for optimization problems where minimizing time is crucial, such as autonomous racing. The best path tracking result was achieved while using the parameters $q_c = 10$ and $q_l = 1$ (Figure 6.14) which would be most useful for situations where precision is the most important factor, e.g, with a demanding path planner. The system was, however, sensitive to small penalties of the angular velocity leading to over steering. This was likely caused by positioning error during high rotational speeds. This was tested and will be discussed in the next section.

The trajectory tracking test (Figure 6.22) resulted in a mean contouring error of $e_c = 4.14$ which is slightly larger than the best LMPCC tracking, and a travel time of 13.5 s which is slower than the fastest LMPCC path following. The travel time can be decreased by adjusting the reference trajectory but it is heuristic, which is the reason why MPCC is preferable. Also, due to the high possible rotation speed of the turtlebot

The one clear limitation of MPCC is again the performance during the sharp turns where the reference functions change. At these points the difference in derivative was quite significant. At these points the contour error was always the largest, since the algorithm calculates false predictions close to the function edge. The solution for this problem would be to make sure that both functions have the same derivative at their meeting point. Another solution is to perform cubic spline interpolation for the target path and formulate θ_k as the arc length of x_c and y_c.

7.3 Real-Time Process Model Validation

In each experiment the robot deviated from the desired path which could indicate an inaccurate model. However, since the position estimations deviated significantly during stationary rotation this might also be caused by faulty measurements. There is no true position from an external source, such as a camera mounted stationary somewhere on the testing ground. One notable result, however, was the experiment when the robot attempted to drive along a straight line after a 180^o turn. The robot began drifting slightly in the direction in which it previously had turned. This was visually confirmed and could therefore not be caused by faulty position estimations. The wheels are not rotating at the same speed, which could indicate a disturbance in the inner velocity loop. This issue, in relation to errors in the AMCL, would explain the deviations in all the experiments.

By testing different initial variance approximations $\alpha_1 - \alpha_4$ the position estimation performance was also explored. Generally the best performance came from larger parameter values. In the real-time MPCC and trajectory tracking experiments we therefore used $\alpha_1 - \alpha_4 = 0.3$ to improve estimation accuracy. For large angular velocities the position estimate still deviated significantly which is likely the reason why the tracking error increased for lower weights on r_{u_2} in the MPCC experiments.

The testing environment could also be a cause for faulty positions, i.e., data coming in to the AMCL is not sufficiently good to determine the true position. For further testing of the MPCC, we improved the accuracy of the position estimator by creating a more distinct mapping area.

8 Conclusion

The application for Model Predictive Contouring Control is almost universal in autonomous driving. Although seeing as in many autonomous contexts maximizing speed is of less importance, MPCC is most efficient during autonomous racing. The linear approximation of MPCC was very accurate and likely viable for on-line driving. The solver FORCES Pro was efficient and proven to be able to solve highly non-linear optimization problems with high speeds. The limitation was that using FORCES Pro in Matlab does not fully take advantage of the high-speed solver and better real-time results would most likely be if the algorithm had been was implemented in the embedded system, using the auto generated C code from the solver. The reference path was followed successfully and can be improved by formulating θ_k as the arc length of both x and y. Using a differential drive also doesn't take full advantage of MPCC because such a process has a large maximum turning speed. The dynamic reference point generation in an MPCC controller would be useful for a car with turning angle limitations.

The main objective of tracking accuracy and minimizing time using MPCC was successful, even with some faulty measurement from the AMCL. As long as the control signals do not change too rapidly the model is an adequate approximation of the real process. This model inaccuracy would need to be modeled for better results as well as improving the position estimation by using a more accurate LIDAR, or an overhead camera.

8.1 Future work

For future work we would like to implement the algorithms in the embedded system using the auto-generated C code from FORCES Pro. We would then port our algorithm to C++ and possibly create a package for it in ROS. This would remove the function call time in Matlab and make it possible to use the non-convex formulation and test its viability for real-time applications. We would also like to change the reference path to smooth cubic spline interpolated functions.

As the purpose of this thesis was autonomous car driving another form of future work would be to use another prototype that could be modelled as a dynamic bicycle model.

As mentioned before a better estimation of the position would be achieved by using a more accurate LIDAR sensor, or a camera mounted above for indoor testing.

In order for this project to be deployed in an urban area, more work needs to be done. Such as turning the city streets into reference functions, so that the contouring control can operate and create control signals according to the desired reference path. It would be an interesting continuation of the work to design a path planner meant for urban driving. For this increased scope the path planner would need to incorporate dynamic obstacle control.

Bibliography

[1] F. Borrelli, A. Bemporad, and M. Morari. *Predictive Control for linear and hybrid systems*. 2015.

[2] S. Boyd and L. Vandenberghe. *Convex Optimization*. MIT Press, 2004. [Chapter 2-4, 11].

[3] A. Domahidi and J. Jerez. FORCES Professional. embotech GmbH (http://embotech.com/FORCES-Pro), July 2014.

[4] A. Domahidi, A. U. Zgraggen, M. N. Zeilinger, M. Morari, and C. N. Jones. Efficient interior point methods for multistage problems arising in receding horizon control.

[5] D. Dougherty and D. Cooper. A practical multiple model adaptive strategy for single-loop mpc, 2002.

[6] Handling road or block corners. ((http://wannadrive.com/handling-road-or-block-corners)). [Online; 14 January, 2017].

[7] T. Faulwasser. Optimization-based solutions to constrained trajectory-tracking and path-following problems. 2003.

[8] T. Glad and L. Ljung. *Reglerteori Flervariabla och Olinjära metoder*. Studentlitteratur, 2011.

[9] G. Grisetti, C. Stachniss, and W. Burgard. Openslam. ((http://openslam.org/gmapping.html)). [Online; 21 December, 2016].

[10] D. Hershberger, D. Gossow, and J. Faust. rviz. ((http://wiki.ros.org/rviz)). [Online; 21 December, 2016].

[11] C. R. Inc. Turtlebot 2. ((https://www.clearpathrobotics.com/turtlebot-2-open-source-robot/)). [Online; 27 December, 2016].

[12] P. Jayakumar, J. L. Stein, and T. Ersal. A multi-stage optimization formulation for mpc-based obstacle avoidance in autonomous vehicles using a lidar sensor, 2014.

[13] F. Kühne, W. F. Lages, and J. M. G. da Silva Jr. Model predictive control of a mobile robot using linearization.

[14] D. Lam, C. Manzie, and M. Good. Model predictive contouring control. 2010.

[15] S. M. LaValle. *PLANNING ALGORITHMS*. Cambridge University Press, 2006. [Chapter 13, page 726-729].

[16] A. Liniger, A. Domahidi, and M. Morari. Optimization-based autonomous racing of 1:43 scale rc cars. 2014.

[17] J. Löfberg. Minimax approaches to robust model predictive control, 2003.

[18] K. Macek, R. Philippsen, and R. Siegwart. Path following for autonomous vehicle navigation based on kinodynamic control. 2009.

[19] J. Maciejowski. *Predictive Control with Constraints*. Prentice Hall, 2001. [Chapter 1-3].

[20] Open Source Robotics Foundation. ROS/Concepts. ((http://wiki.ros.org/ROS/Concepts)). [Online; 21 December, 2016].

[21] Scania. Autonomous Transport systems 2016. ((https://www.scania.com/group/en/section/pressroom/backgrounders/ autonomous-transport-systems-2016/)). [Online; 16 January, 2017].

[22] Dieter Fox. Kld-sampling: Adaptive particle filters. ((https://articles.nips.cc/article/1998-kld-sampling-adaptive-particle-filters.pdf)), 1998. [Online; 18 December, 2016].

[23] S. Thrun, W. Bulgard, and D. Fox. *Probabilistic Robotics*. MIT Press, 2005. [Chapter 1-2, 4-6, 8].

[24] Volvo. Autonomous Driving. ((http://www.volvocars.com/intl/about/our-innovation-brands/intellisafe/autonomous-driving)). [Online; 19 January, 2017].

[25] H. Wang, J. Kearney, and K. Atkinson. Arc-length parameterized spline curves for real-time simulation. 2014.

Lund University **Department of Automatic Control** **Box 118** **SE-221 00 Lund Sweden**	*Document name* MASTER´S THESIS *Date of issue* April 2017 *Document Number* ISRN LUTFD2/TFRT--6029--SE
Author(s) Johan Kellerth Fredlund Kenan Sadik Sulejmanovic	*Supervisor* Michael Bano, Uniti Bo Bernhardsson, Dept. of Automatic Control, Lund University, Sweden Anders Robertsson, Dept. of Automatic Control, Lund University, Sweden Karl-Erik Årzen, Dept. of Automatic Control, Lund University, Sweden (examiner) *Sponsoring organization*

Title and subtitle

Autonomous driving using Model Predictive Control methods

Abstract

This thesis explored the path following applications of autonomous driving, where the purpose is to sense the environment and navigate to a goal without human intervention. Autonomous driving enables safer journeys by removing human error in driving, as well as reducing fuel consumption and driving time by optimizing the engine and brake actuation. In 2017 it is a well researched topic in the academic and industrial world.

The scope of this work was to create a reliable and efficient path following solution which enabled a robot car to accurately traverse along any given path. The objective was therefore to minimize path error with the additional objective to minimize the traversal time.

The method used to realize the objective statement was Model Predictive Contouring Control (MPCC). MPCC uses optimization based prediction to smoothly control the car along a reference path, where actuator constraints and the objective are both handled in the optimization problem. The reference was also defined as a geometric function instead of a specified trajectory of coordinates. With the addition of objective weights it was possible to choose the relative importance of the two objectives, path accuracy and minimizing time. The resulting problem formulation, which was heavily nonlinear, was linearized to reduce computation time and solved using FORCES Pro. Simulations where made in Simulink, while real-time execution of the MPCC were achieved with a robot prototype, Robotics Operating System (ROS) and Matlab.

The result suggested that MPCC was able to operate efficiently in a real time environment. Manipulating the objective weights resulted in predictable operation, which makes it possible for a user to control whether faster traversal or tracking performance is desired. It can be concluded that MPCC is a viable control method for autonomous driving, in particular for autonomous racing purposes. Additionally, the simulation results regarding the non-linear MPCC showed a surprisingly fast solve time, which suggest that it would be feasible for on-line driving using the solver FORCES Pro.

Keywords		
Classification system and/or index terms (if any)		
Supplementary bibliographical information		
ISSN and key title 0280-5316		*ISBN*
Language English	*Number of pages* 1-79	*Recipient's notes*
Security classification		

http://www.control.lth.se/publications/