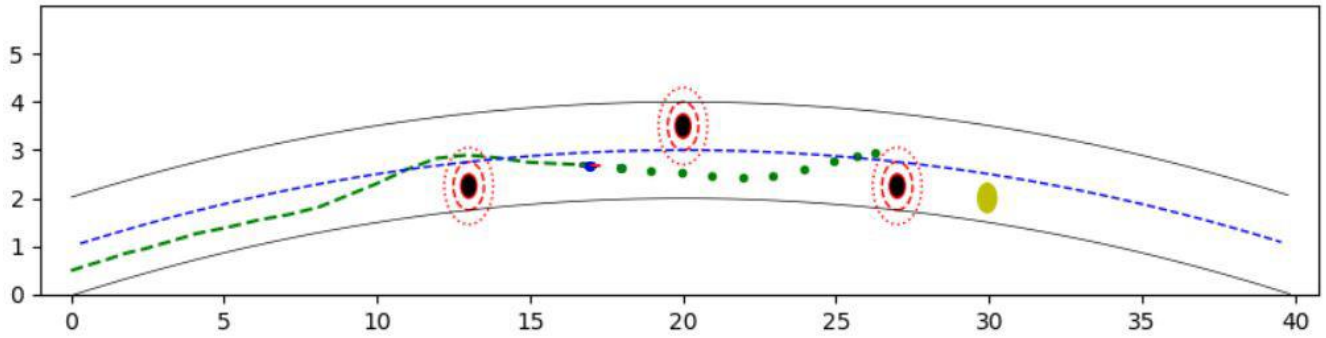

Evaluation of optimization algorithms in MPC applications

Maneuvering a differential drive robot in simulations and hardware

Master's thesis in Systems, Control and Mechatronics

Mattias Gerle
Simon Johansson

DEPARTMENT OF ELECTRICAL ENGINEERING

CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2020
www.chalmers.se

Master's thesis 2020

Evaluation of optimization algorithms in MPC applications

Maneuvering a differential drive robot in simulations and hardware

Mattias Gerle & Simon Johansson

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2020

Evaluation of optimization algorithms in MPC applications Maneuvering a differential drive robot in simulations and hardware
Mattias Gerle & Simon Johansson

Supervisor: Bengt Lennartson, Department of Electrical Engineering
Examiner: Bengt Lennartson, Department of Electrical Engineering

Master's Thesis 2020:NN
Department of Electrical Engineering
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Differential drive robot manuvering around three obstacles using a MPC controller.

Typeset in LaTeX
Gothenburg, Sweden 2020

Abstract

This thesis aims to address and evaluate the MPC approach for the control of a small differential drive robot in static and dynamic environments. Three different optimization algorithms where evaluated on three different simulated scenarios. The algorithms are *Sequential Quadratic Programming* (SQP), *Interior-Point Method* (IP) and *Proximal Averaged Newton-type method for Optimal Control* (PANOC). For collision avoidance a variant of the collision cone method was used, namely a nonlinear collision cone. PANOC proved to be the preferable algorithm due to it being more robust and it's superior speed. Thus, PANOC was implemented into hardware for further evaluation. The hardware implementation was done using an Arduino robot, controlled via bluetooth, and a camera attached to the ceiling. The hardware implementation came with it's own set of challenges. Using Open Source Computer Vision Library (OpenCV) the ego vehicles state space can be identified. To get a better estimate of the robots position an Extended Kalman Filter was used. All the optimizers managed to drive all the scenarios tested but performed differently. PANOC showed a lot of promise in the field of non-linear optimization. It also successfully drove a robot through a simple environment.

Acknowledgement

Firstly, we want to thank professor Bengt Lennartson at Chalmers for his guidance and help as examiner in this thesis.

Secondly, we want to give a special thanks to Tobias Olsson and Combine for their guidance, help and support to be able to fulfill this thesis.

Thirdly, we also want to thank Sabino Roselli for his help and for sparking our interest in the PANOC algorithm, which a large part of this thesis is based on.

Mattias Gerle & Simon Johansson, Gothenburg, June 2020

Contents

Nomenclature

Constants

μ	Barrier parameter
k	Discrete time step
N	Prediction horizon
n	n:th step in the prediction horizon
Q_S	State stage penalty matrix
Q_T	State terminal penalty matrix
R	Input penalty matrix
T_s	Sampling time
q_r	Reference state vector
u_r	Reference input vector

Other symbols

c_{eq}	Nonlinear equality constraint function
c_{in}	Nonlinear inequality constraint function
$cost$	Variable dependent cost function
g	Relaxed, non-smooth, non-convex constraint function in PANOC
h	Combined and smoothed cost function in PANOC
i	Iteration index
j	Iteration index
prox	Proximal mapping operator

Variables

$\hat{u}$	Minimizer for smooth part of a function, used in PANOC
λ	Lagrange multiplier
$\tilde{q}$	Error state vector including $x - x_r$, $y - y_r$ and $\theta - \theta_r$ for the different stages in the predicition horizon
s	Slack variable
x	Decision variable in optimization
$\tilde{u}$	Error input vector including $v - v_r$ and $\omega - \omega_r$ for the different stages in the predicition horizon
q	State vector including x, y and θ for the different stages in the predicition horizon
u	Input vector including v and ω for the different stages in the predicition horizon

Acronyms

AD Automatic backward Differentiation.

ALM Augmented Lagrangian Method.

BFGS Broyden–Fletcher–Goldfarb–Shanno.

EKF Extended Kalman Filter.

FBE Forward-Backward Envelope.

FBS Forward-Backward Splitting.

IP Interior-Point.

KKT Karush-Kuhn-Tucker.

L-BFGS Limited-memory Broyden–Fletcher–Goldfarb–Shanno.

MPC Model Predictive Control.

NMPC Nonlinear Model Predictive Control.

PANOC Proximal Averaged Newton-type method for Optimal Control.

PM Penalty Method.

RRT Rapidly exploring Random Tree.

SQP Sequential Quadratic Programing.

Glossary

Convergence criteria Criteria that tells the optimizer to stop since it has converged.

Decision variables The variables to be optimized in the optimizer.

Ego vehicle The vehicle which is to be controlled.

Forward backward envelope Merit function used in PANOC.

Forward backward splitting Also known as the proximal gradient method which is a generalization of the projected gradient method.

Global convergence The solution has converged to a global minimum.

Local convergence The solution has converged to a local minimum.

Merit function Function that simulate the objective function and adjust the step size for better convergence.

Newton's method Common optimization algorithm.

Objective function Function to minimize.

Oracle A model from which one can query information about the objective function, such as function values and gradients.

Stopping criteria Criteria that has to be fulfilled for the optimization algorithm to stop.

1 Introduction

The vast field of autonomous vehicles is expanding rapidly. The field contributes with a lot of interesting and challenging problems. One such problem is how an autonomous vehicle can make informed decisions based on the environment. This sparks the interest for Model Predictive Control (MPC) since it can implement model and environment constraints. However, due to the computational complexity the MPC can be too demanding to run on real time systems, which makes it paramount to lower computation speeds while not sacrificing performance. The added complexity also reduces the possibility for convergence and therefore the quality of control input. This thesis aims to provide viability for MPC for path planning in real-time applications in complex environments with a realistic point of view.

1.1 Background

Autonomous vehicles have in recent years gained much attention and are becoming more and more common in both the industrial and commercial world. There is therefore a lot of financial interest to improve these methods. As this field develops and autonomous vehicles becomes more common the need for safe and robust methods increases. To increase the safety a deep understanding of the algorithms used in decision making must be acquired. In order to improve and find optimal methods for autonomous drive a number of difficult problems must be addressed. One of which is path planning. The path planning is the core of the problem as it addresses many of the risks and also the success of driving from A to B. Often the autonomous drive algorithms must be able to handle plenty of different constraints such as collision avoidance, time limits and the systems physical constraints. All these constraints shrinks the feasible space of viable paths. Further, in real life things are most often nonlinear which brings it's own set of difficulties. In this thesis the environment and system constraints are nonlinear. If the environment in which the vehicle operates is changing dynamically the task becomes even harder as the applied constraints change and predicting the future state of the environment becomes vital. One wants to guarantee that the vehicle performs within the given constraints while also optimizing some factors such as for example fuel consumption, jerk or speed. An interesting method, which has a lot of potential is *Model Predictive Control* (MPC). The MPC method uses an optimization algorithm to find the optimal control input with respect to the given system over a period of time. To be able to do this online would present new opportunities for MPC.

At the core of the method lies the optimization algorithm which is responsible for planning the path whilst respecting the systems limitations with the use of constraints. However, optimization algorithms is a vast field in itself and determining how different algorithms work with respect to each other in different conditions is hard to determine by merely looking at a problem due to their potential size and complexity. An optimization algorithm is sometimes thought of as a black box as the problems can very quickly become huge, very nonlinear, non-smooth and complex [1]. There is therefore a need for understanding and comparing these different algorithms to see what their strengths and weaknesses are.

1.2 Related work

There are multiple methods for solving the path planning problem of robots. Well known and used methods are among others Rapidly exploring Random Tree (RRT), RRT* and RRT*-Smart [2]. They are quite low on computation power and some of them can quite easily achieve near optimal paths. However, these algorithms generally don't take the vehicles dynamics into consideration, somewhat limiting their real life application. Although they could work well for robots with simple dynamics since these could turn

on the spot, more complex models such as car like models could render the results useless. Extensions of the RRT algorithms to include kinematics has been done using a kinodynamic version of RRT [3]. This would make the results more usable, however this comes at the cost of more computational complexity, which currently does not allow it to be used in online applications. Other projects uses a MPC instead. In the MPC it's easy to implement the vehicle models, and it can quite easily be manipulated by using costs and constraints since it works like an optimization problem. For this reason many projects choose to uses MPC for trajectory planing[4], compared to RRT methods. However, as for the RRT methods, MPC can be computationally expensive.

As previously mentioned, it's well known that MPC's can be computationally expensive, thus even faster optimization algorithms that can perform with good accuracy is needed. This makes advancements in optimization of utmost priority. There are a few classic and well established algorithms like Sequential Quadratic Programming (SQP) and Interior-Point (IP) [1], but also a newer algorithm called Proximal Averaged Newton-type method for Optimal Control (PANOC) [5]. Linear MPCs of today can perform with high speeds, making it possible to implement them in embedded systems. However, the same is not obvious for nonlinear systems. This sparked an interest for fast solvers such as PANOC, since it allows for high speed computations of nonlinear systems. Among other things PANOC has been shown to work on nonlinear systems such as a mobile robot with a trailer [6], but this is merely a conceptual test. The company Embotech has with their IP solver made an actual implementation into a car. They show how it can effectively avoid a suddenly appearing obstacle. They also show how a simulated car can drive on a road without obstacles [7]. Even tough they have accomplished astonishing things the question arises, what is the limitations for using MPC in self driving vehicles?

One of the main challenges in MPC is how to model the environment. Where, in vehicle applications, obstacle avoidance is an especially hard part. This is due to that many scenarios infers highly non-linear and computationally expensive operations. It is also one of the most important constraints which can not be broken. There exists multiple methods for how the MPC controller could identify obstacles and act accordingly. Preferably the obstacles will be included as constraint in the MPC formulation. In *Path Planning for Autonomous Vehicles using Model Predictive Control* [8] obstacles are formulated as linear polygons and are supplied to the optimizer. This is a unique solution where one simply eliminate the obstacle in the subspace of feasible solutions. However, with this solution the optimizer is not guided on how to avoid the obstacle. It could drive dangerously close towards the obstacle before it takes action and further this solution does not guarantee a collision free path in between time steps. Another type of modelling that allows the MPC to react in a more human-like fashion is the so called Collision Cone (CC). The Collision Cone is a method that is using the position and relative velocity of the obstacles to constrain the directions in which a collision is achieved, thus making the vehicle take precautions long before reaching the obstacle[9][10].

1.3 Our contributions

This thesis aims to answer and analyse the following research questions:

- *How do state of the art optimizers perform in realistic, highly constrained, non-linear and non-smooth environment?*
  - *What are the reason for the differences and in which applications should they all be used?*
- *Could MPC be used to navigate a vehicle in a general environment?*

– *To what extent must the problem be limited for MPC to work and what is its potential in the area of self driving vehicles?*

This thesis gives light to how state of the art optimizers actually perform in realistic driving scenarios. It pushes them to the limit to identify when and why they break. It shows some of their limitations, flaws and strengths in different environments and points to improvements that ought to be done. It discusses in which applications they should be used and why. It aims to further validate the use of MPC for real time applications and explore the use of MPC as an online controller for nonlinear systems.

1.4 Problem architecture

The system will first be simulated using a MPC with a differential drive model and later the MPC will be implemented into hardware on a small Arduino robot. The project can be broken down into three main subproblems:

Environmental modeling Given an environment there will exist a number of constraints modeling the surroundings in which the vehicle will navigate. Those things could be other cars, stationary obstacles or a road it must stay on. The places the car can not drive on the so called unfeasible subspace must be modeled with constraints. One of the big problems is how to implement obstacles as constraints and how to avoid collision with these. The constraints must further be set up in such a way that the optimizer can converge to a good path in the environment.

Model constraints The computed path and the corresponding control inputs must follow the limitations of the Arduino robot and the limitations of the model which describes it. Further, the control inputs must be bounded so that the system is behaving in a way that corresponds well to reality, as example there is a limit to how fast a set of motors can rotate and this needs to be present in the MPC. All of these different constraints will limit the feasible subspace of solutions.

In hardware, estimating the state of the vehicle In the simulations the position will be updated according to the differential drive model and are therefore assumed to be completely and accurately defined. For testing on the real robot the position must be estimated quite well. If the position of the vehicle is not well estimated this can lead to a violation of the constraints. The vehicle's state will be identified using a camera in the ceiling, however, the camera will be victim to noise, which makes it impossible to view the measurements as completely accurate. On the other hand the Arduino robot is not an exact replica of the model and the motors are simple brushed DC electric motors, meaning that there are a lot of disturbances impacting the physical system as well. Thus the model and camera measurements will be balanced using a non-linear Kalman filter.

1.5 Limitations

The project will only evaluate SQP, IP and PANOC as algorithms. The two former ones are widely used algorithms and are common choices for nonlinear MPC [1]. PANOC is a new method proposed by L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos [5] and made into a open source optimizer by P. Sopasakis and E. Fresk and P. Patrinos [11]. The project will not go into depth about the optimizers, but will present the general idea of how the algorithms work. Focus will not be put on evaluating the differences and why they perform differently theoretically, but rather the performance of simulated experiments. Said experiments will include a vehicle navigating through three different predefined scenarios.

In other words this study is not aiming to benchmark the different optimizers. This is due to the fact that benchmarking optimizers and comparing them is a big field in off itself where specific benchmarking scenarios are presented and tested. These tests are made in clean environments, without disturbances and outer influences in order to achieve comparable results. An example of such a test scenario is the classic Rosenbrock function [12], which is used to evaluate gradient based algorithms. Due to these facts it's not desirable to benchmark optimization algorithms on MPCs.

Further, it's important to note that PANOC is implemented in Python. This is due to the fact that the Python version is more developed than the Matlab version. This however won't effect much as the PANOC API is written in Rust and only time efficient operations are made in Python[11]. SQP and IP are implemented through Matlab, thus the language's inherent weaknesses and strengths might influence the algorithms performance somewhat, but this effect compared to the structural differences in the algorithms are assumed to be negligible. Any time related evaluations will strictly be performed over the call to the optimization algorithm in order to minimize the influence of the different languages.

2 Theory

Firstly in this chapter a general explanation of how a MPC controller works will be presented. The fundamental part of the MPC controller is the optimization algorithm, thus what follows is a presentation of the relevant algorithms for this thesis. The first two optimizers are SQP and IP, which are considered two of the most powerful and well used non-linear optimization algorithms[1]. The last optimizer explained, PANOC, is comparatively new and according to the creators suitable for embedded systems due to its fast convergence rate.

2.1 General Model Predictive Control

The MPC controller uses a system model to predict where the system will end up with a given control input. The model can be used iteratively, given some initial starting point, to allow predictions further into the future. Consider a linear discrete state space model

$$q_{k+1} = Aq_k + Bu_k,\ k = 1, 2, .., N \tag{1}$$

where q is state vector and u is an input vector. Using this iteratively for N time steps generates information about where the system will be at time step N. Nonlinear systems will from here on be of specific interest, since the system at hand will be nonlinear. Nonlinear models alters the formulation to

$$q_{k+1} = f_{model}(q_k, u_k),\ k = 1, 2, .., N \tag{2}$$

Time step N is also known as the *prediction horizon* and denotes how far into the future the MPC controller makes predictions. Now given some reference state q_r and reference input u_r the MPC will supply the system with control inputs u_k, which is done by using some optimization algorithm. The type of algorithm varies depending on if the problem is linear, quadratic or nonlinear. The MPC supplies a control input by formulating an optimization problem in a suitable manner.

$$\min_{u_0, u_1, \ldots, u_{N-1}} \sum_{k=0}^{N-1} (\tilde{q}_k^T Q_S \tilde{q}_k + \tilde{u}_k^T R \tilde{u}_k) + \tilde{q}_N^T Q_T \tilde{q}_N \tag{3a}$$

$$\text{s.t.} \quad q_{k+1} = f_{model}(q_k, u_k) \tag{3b}$$

where $\tilde{q}_k = q_k - q_r$ and $\tilde{u}_k = u_k - u_r$ and minimizing this error drives the decision variables towards the references. Each step along the prediction horizon implies an added cost as $\tilde{q}_k^T Q_S \tilde{q}_k$, this cost is better known as stage cost. Further, in (3a) Q_S, Q_T and R are user chosen weight matrices, which allows the user to choose which state that is of most importance to minimize. With (3) it's possible to use an optimization algorithm to find the values of u that would optimally drive the system to the reference state. A chosen number of consecutive values from the u vector, also known as the *control horizon*, is then applied on the real system which moves accordingly and the MPC is reinitialized. [13]

Depending on how the MPC formulation is designed the complexity of the problem is determined. Highly nonlinear and constrained problems are harder for an optimizer to solve compared to, as example, a quadratic optimization problem with linear constraints. A longer prediction horizon also increase the computational complexity, since it increases the amount of decision variables that is included in the objective function and is subject to constraints. Thus, increasing the prediction horizon can increase the performance of the system, but it may also dramatically increase the computational cost or decrease the

probability of global convergence. More constraints can also be added. The model could be improved upon by adding more constraints modeling the capabileties of the system in a more realistic way. The environment in which the vehicle drives in could also be modeled with constraints which are added in a similar fashion.

There is also something to be said about hard and soft constraints. Hard constraints are limiting the feasible space of the optimizer, making some states or inputs unusable. Soft constraints, on the other hand, are formulated into the objective function and simply penalizes the optimizer for taking actions violating the constraint.

2.2 Sequential Quadratic Programming

One of the most common and proven methods for nonlinear constrained optimization is the Sequential Quadratic Programming (SQP) method. This method is widely used and one of the best methods for non-linear optimization. It is an so called active set method which means that it only works with the constraints that are currently active in the optimization. It typically works well in environments that has a lot of constraints as it is guided by these towards the solution. This algorithm is thoroughly explained by Nocedal, J. and Wright, S. in *Numerical Optimization* [1]. Note that there are many different versions of SQP and only a general conceptual explanation is given below. SQP works by sequentially solving relatively simple quadratic sub-problems of the main problem until convergence. Assume the following minimization problem is to be solved with SQP:

$$\min_{x} \quad f(x) \tag{4a}$$

$$\text{s.t.} \quad c_{eq}(x) = 0 \tag{4b}$$

The SQP method aims to set up and solve the Karush-Kuhn-Tucker (KKT) conditions for the minimization problem above using Newton's method. For this the jacobian of the constraints are needed and are written as:

$$\nabla c_{eq}(x)^T = [\nabla c_{eq,1}(x), \nabla c_{eq,2}(x), ..., \nabla c_{eq,m}(x)] \tag{5}$$

Now the quadratic sub-problem seen in (6) is to be solved to find the step d_i. The terms in (6) are approximated and together they form a quadratic minimization problem. One of the main problems with SQP comes from estimating the Hessian $\nabla^2_{xx}\mathcal{L}$, where $\mathcal{L}(x, \lambda) = f(x) - \lambda^T c_{eq}(x)$. Note that 6a is a Taylor series that represent the function value f_{i+1}. Every sub-problem is subject to the same constraints as in the original problem, but as SQP is an active set method the set of constraints accounted for varies.

$$\min_{d} \quad f_i + \nabla f_i^T d_i + \frac{1}{2} d_i^T \nabla^2_{xx}\mathcal{L}_i d_i \tag{6a}$$

$$\text{s.t.} \quad \nabla c_{eq,j}(x_i)^T d_i + c_{eq,j}(x_i) = 0, \quad j = 1, 2, ..., m \tag{6b}$$

The Lagrange multipliers are updated by solving system (7), where (7a) ensures that the KKT condition *no feasible decent* is met and (7b) ensures that the constraint conditions are met through newtons method.

$$\nabla^2_{xx}\mathcal{L}_i d_i + \nabla f_i - \nabla c^T_{eqi} \lambda_{i+1} = 0 \tag{7a}$$

$$\nabla c_{eq,i} d_i + c_{eq,i} = 0 \tag{7b}$$

If the SQP problem is constrained, the algorithm should use a merit function to determine if the computed step length is acceptable or not. This is needed since (6) does not necessarily share the same minimum as the original problem. There are plenty of different merit functions that could be used for these types of problems. Matlab's *fmincon* uses a merit function formed by Han's [14] and Powell's findings [15].[16] Simply put, the merit function finds the value α_i, which changes the step update to:

$$x_{i+1} = x_i + \alpha_i d_i \tag{8}$$

The merit function can therefore help the algoritm to adjust the step so that better convergence is achieved. If x_{i+1} satisfies the convergence test the algorithm stops otherwise the terms of 6a are evaluated again and a new sub problem is solved to generate a new iterate.

SQP methods are only efficient when the Hessian $\nabla^2_{xx}\mathcal{L}$ is guaranteed to be positive definite which occurs when the quadratic model has a well defined strong minimizer. If this is not the case then there exist two main ways to tackle this problem. One could enforce a trust-region where one thinks that the new iterate is well defined and useful by restricting the step d, $||d|| \leq \Delta_i$. If the step is found to be too big, Δ_i is shrunk and new calculations are made. The inclusion of this new constraint may cause the problem to become infeasible and handling these situations are complicated and computationally expensive. The other well accepted practise is to modify the Hessian until it becomes positive definite. A popular method is to approximate the Hessian with a Quasi-Newton Method BFGS, (named after its inventors, Broyden, Fletcher, Goldfarb, and Shanno) to ensure that d_i is in a descent direction of the objective function. This can however introduce distortions in the model. There is no clear winner of the former Trust-Region approach or the latter when the Quasi-Newton method simply modify the Hessian. Matlab uses the Quasi-Newton technique in their *fmincon* function [16].
With the SQP framework presented above it can easily be extended for more general nonlinear problems including also inequality constraints. Simply include the inequality constraints and linearize them in the same manner as the equality constraints in (5).

It is very common to relax the constraints by penalties to avoid the risk of the problem getting infeasible. The quadratic sub problem is penalised as

$$\min_{x,s_1,s_2,s_3} \quad f(x) + \mu \sum_{i=0}^{N_{eq}} (s_{1i} + s_{2i}) + \mu \sum_{i=0}^{N_{in}} s_{3i} \tag{9a}$$

$$\text{s.t.} \quad c_{eq}(x_i) = s_{1i} - s_{2i} \tag{9b}$$

$$c_{in}(x_i) \geq -s_{3i} \tag{9c}$$

$$s_{1i}, s_{2i}, s_{3i} \leq 0 \tag{9d}$$

Where N_{in} is the number of inequality constraints and N_{eq} is the number of equality constraints. Also, s_{1i}, s_{2i} and s_{3i} are slack variables. This brief introduction to SQP methods is supposed to give the reader an intuition about how the algorithm operates. As mentioned earlier evaluating the Hessian is something that may take a long time to approximate in SQP algorithms. This is partly since it needs to be positive definite which means that reevaluation or alterations may be needed to ensure this criteria. SQP is an active set method so every time the active set is changed the Hessian needs to be recalculated since the original optimization problem changes and the Hessian with it. [1].

2.3 Interior-Point

Interior point (IP) algorithms are also known as barrier method algorithms and are along with SQP the most used algorithms for non-linear optimization problems. As the name implies, it finds a solution by traversing in the interior of the problem's feasible space. It therefore operates very differently from SQP as it works far from the constraints. It typically works good on problems that are not constrained much since it hinders the algorithm or increases the risk of it becoming ill conditioned. There are multiple variants of Interior-Point methods, which comes with their own strengths and weaknesses. Once again using the work of Nocedal, J. and Wright, S. [1] a basic version of the IP algorithm is presented to get some insight in how it operates. Given a nonlinear optimization problem of the form:

$$\min_{x,s} \quad f(x) \tag{10a}$$
$$\text{s.t.} \quad c_{eq}(x) = 0 \tag{10b}$$
$$c_{in}(x) - s = 0 \tag{10c}$$
$$s \geq 0 \tag{10d}$$

Note that the inequality constraints, c_{in}, have been rewritten as equality constraints with the use of a slack variable s. The inequality constraint is eliminated by adding it to the objective function as a barrier penalty.

$$\min_{x,s} \quad f(x) - \mu \sum_{i=1}^{N_{in}} log(s_i) \tag{11a}$$
$$\text{s.t.} \quad c_{eq}(x) = 0 \tag{11b}$$
$$c_{in}(x) - s = 0 \tag{11c}$$

Here the lower bound on s has been included in the term $-\mu \sum_{i=1}^{m} log(s_i)$ since the log-term bounds s from below. Note that $log(s)$ approaches ∞ as s approaches 0. This can be visualized as a barrier positioned at the inequality constraint, thus giving the IP method the alternative name *Barrier method*. The barrier parameter μ is chosen as a positive parameter which determines the strength of said barrier. How this parameter is supposed to be chosen is under subject of discussion, one valid approach is to update it with some step size during each iteration.

Although it's possible to use the formulation presented in (11) straight away it can present alot of nonlinearities. Thus it's preferable to find a more linear form. This is done by finding the problems KKT conditions and solve these instead. The formulation in (11) have the corresponding KKT conditions:

$$\nabla_x f(x) - \nabla_x c_{eq}(x)^T y - \nabla_x c_{in}(x)^T z = 0 \tag{12a}$$
$$-\mu S^{-1} e + z = 0 \tag{12b}$$
$$c_{eq}(x) = 0 \tag{12c}$$
$$c_{in}(x) - s = 0 \tag{12d}$$

Note that $\mathcal{L}(x, s, y, z) = f(x) - c_{eq}(x)^T y - (c_{in}(x) - s)^T z$, which implies that $\nabla_x \mathcal{L}(x, s, y, z) = \nabla_x f(x) - \nabla_x c_{eq}(x)^T y - \nabla_x c_{in}(x)^T z$. Here y and z are the corresponding Lagrange multipliers and S is an matrix containing all slack variables $(s_1, s_2, s_3, \dots)$ on its diagonal. Further, e is a vector of ones to satisfy the

vectorization. The KKT conditions are necessary for the found solution x^* to be a local optimum, and thus it's preferable to solve this problem instead of (11). With the use of Newton's method on each equation in system (12) the following is achieved:

$$\begin{bmatrix} \nabla^2_{xx}\mathcal{L} & 0 & -\nabla c^T_{eq}(x) & -\nabla c^T_{in}(x) \\ 0 & Z & 0 & S \\ \nabla c_{eq}(x) & 0 & 0 & 0 \\ \nabla c_{in}(x) & -I & 0 & 0 \end{bmatrix} \begin{bmatrix} d_x \\ d_s \\ d_y \\ d_z \end{bmatrix} = - \begin{bmatrix} \nabla f(x) - \nabla c^T_{eq}(x)y - \nabla c^T_{in}(x)z \\ Sz - \mu e \\ c_{eq}(x) \\ c_{in}(x) - s \end{bmatrix} \tag{13}$$

With this the Newton directions d_x, d_y, d_s and d_z can be computed, and now it's possible to take a step along the Newton direction via:

$$x^+ = x + \alpha^{max}_s d_x \tag{14a}$$
$$s^+ = s + \alpha^{max}_s d_s \tag{14b}$$
$$y^+ = y + \alpha^{max}_z d_y \tag{14c}$$
$$z^+ = z + \alpha^{max}_z d_z \tag{14d}$$

Here α^{max}_z and α^{max}_s determines how big the step length along the search direction should be. To find α we need to follow the *fraction to the boundary rule*. Which is a way of guaranteeing that the variables s and z aren't approaching their lower bounds too fast. If they are not bounded away from zero the second row in the matrix in (13) can lose rank and the problem becomes ill conditioned.

$$\alpha^{max}_z = max\{\alpha \in (0,1] : z + \alpha d_z \geq (1-\tau)z\} \tag{15a}$$
$$\alpha^{max}_s = max\{\alpha \in (0,1] : s + \alpha d_s \geq (1-\tau)s\} \tag{15b}$$

Where $\tau \in (0,1)$. The new found iterate must also decrease a chosen merit function to be accepted. If x_{k+1} satisfies the convergence test the algorithm stops, otherwise the terms of (13) are evaluated again and the linear system is solved to generate a new iterate.

This brief introduction to IP methods is supposed to give the reader an intuition about how the algorithm operates. One advantage of IP over SQP is that it scales well with the number of optimization variables. On the other hand it scales worse with respect to the number of constraints.[1]

2.4 Proximal Averaged Newton-type method for Optimal Control

The following description is based on the work of L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos [5]. PANOC is a very fast optimizer that requires very little memory to run, which is why it is suitable for embedded devices with limited computation power. It's also promising very high speeds compared to other nonlinear solvers. In short, it combines a proximal operator with a quasi-newton method to converge to an optimal solution. This is a new way of tackling optimization by combining two widely accepted and studied methods for convergence. The authors of the algorithm claim that it is suitable for MPC and that it outperforms the previous described optimizers IP and SQP in the area of nonlinear optimization in embedded devices. Consider the optimization in (16), where l_n and l_N are smooth stage and terminal cost. The function f_n are smooth mappings representing system dynamics. The function g_n represents the relaxed constraints in form of penalties which can be nonconvex, nonsmooth functions.

$$\min_{x} \quad \sum_{i=0}^{N-1} (l_i(x_i) + g_i(x_i)) + l_N(x_N) \tag{16a}$$

$$\text{s.t.} \quad x_{i+1} = f_i(x_i), \quad i = 0, ..., N-1 \tag{16b}$$

Again here N is the prediction horizon. To converge, the step direction and length of the step must be decided, but (13) includes parts which are not differentiable and thus the step cannot be computed. However, the step for the smooth parts can be computed. Let $h(u)$ represent the combined smoothed functions from the stage and terminal costs. If this step $\hat{u}$ is calculated then it can be projected onto the feasible set defined by g. This method is used in the proximal operator which PANOC uses and is partly named after. Evaluation of the proximal mapping of the nonsmooth penalty g, which usually has a very simple closed-form and is written

$$\hat{x}_{i+1} = x_i - \alpha \nabla h(x_i) \tag{17}$$

$$x_{i+1} = prox_{\alpha g}\big(\hat{u}_{i+1}\big) \tag{18}$$

where $prox_{\alpha g}$ is the proximal operator, making a proximal mapping of g and using the step size α.

$$\text{prox}_{\alpha g}(\hat{u}) = \underset{u \in \mathcal{R}^{N_n \hat{u}}}{\arg\min} \big(g(u) + \frac{1}{2\alpha}||u - \hat{u}||^2\big) \tag{19}$$

Again the proximal operator is trying to reduce the value of $g(u)$ while also minimizing a square penalty on the distance from the current iterate $\hat{u}_{k+1}$. In other words the new value x_{k+1} shall be as close as possible to the value that minimize the smooth functions in $\hat{u}_{k+1}$, but at the same time be a good minimizer for the nonsmooth function $g(u)$. In the PANOC algorithm the proximal mapping is used as:

$$\hat{u}_{i+1} = x_i - \alpha \nabla h(x_i) \tag{20a}$$

$$x_{i+1} = \text{prox}_{\alpha g}(\hat{u}_{i+1}) = \underset{u \in \mathcal{R}^{N_n \hat{u}}}{\arg\min} \big(g(u) + \frac{1}{2\alpha}||u - \hat{u}_{i+1}||^2\big) \tag{20b}$$

The input to the prox operator is the gradient of the smooth and convex function. This means that the proximal mapping is penalized for straying to far from the gradient descent evaluation, however the contribution from $g(u)$ must be kept low in order for a new iterate to be found. Thus, the proximal gradient method effectively finds a minimizer for the problem even though it may contain non-smooth parts.

Evaluation of the gradient of the smooth functions h are made using Automatic backward Differentiation (AD). AD is a popular method for calculating the derivatives of a function. The algorithm PANOC uses an API called CasADi to calculate approximations of Jacobians and gradients. This API is especially good for nonlinear problems such as nonlinear MPC [17].

If L is the Lipschitz constant for the minimization problem in (16), then the projected gradient method is guaranteed to converge if $\alpha < 2L$, thus α is chosen accordingly. It will converge to points that are fixed

points (u^*) of the prox operator in (20). In other words at least a local solution to the problem.

These techniques alone converges with at most Q-linear speed, but often slower and is very sensitive to ill conditioning. The creators of PANOC have therefore made it more robust and faster with two main tricks that have shown to be much faster than other forward-backward splitting (FBS) type methods.[5] They address the minimization problem indirectly by minimizing the residual from the current iterate u and the fixed points u^* of the problem. The residual r is calculated with the following operator R.

$$r = R_\alpha(u) = \frac{1}{\alpha}(u - \text{prox}_{\alpha g}(u - \alpha \nabla h(u))) \tag{21}$$

One would then take a step in the direction that minimizes this residual which can be done with the quasi-Newton method L-BFGS.

$$x_{i+1} = x_i - H_i R_\alpha(x_i) \tag{22}$$

where H_i are invertible linear operators that satisfy the secant condition $s_i = H_i y_i$ with $s_i = x_{i+1} - x_i$ and $y_i = R_\alpha(x_{i+1}) R_\alpha(u)$. L-BFGS stores only a few vectors, s and y that represent an approximation of H which makes it very memory efficient and thus suitable for embedded systems. Ideally H_i capture curvature information of R_α and enable superlinear or quadratic convergence when close enough to a solution.

The next big problem to handle is ensuring global convergence since L-BFGS is only convergent in a neighborhood of u^*. To ensure convergence the step size in (22) must be accurately decided. How this is enforced is by using a so called forward-backward envelope (FBE), which is a real valued, continuous merit function that shares the same local/strong minima for $0 < \alpha < \frac{1}{L}$ with the original minimization problem. Using the FBE the problems turns into an unconstrained minimization problem. As L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos [5] shows the problem in (16) converges globally if it is a so called Kurdyka-Łojasiewicz function which is satisfied by all subanalytic functions. This is a mild property.

$$\rho_\alpha(u) = f(u) - \frac{\alpha}{2}||\nabla f(u)^|||^2 + dist_U^2(u - \alpha \nabla f(u)) \tag{23}$$

The function $\rho_\alpha(u)$ describes the FBE merit function where $dist(\beta) =_{u \in U} ||u - \beta||$ is the closest distance to the set U. The calculation of the FBE is based on the same operations as the FBS making it similarly efficient provided the distance to U is easy to compute.

Finally, the inner problem which only involve cheap operators becomes:

$$x_{i+1} = x_i + \gamma_i(-H_i R_\alpha(x_i) + (1 - \gamma_i)\alpha R_\alpha(x_i) \tag{24}$$

where α_i is chosen with help of the merit function $\rho_\alpha(u)$.

To ensure that the constraints are met in the inner procedure described above either the classic penalty method (PM) or augmented Lagrangian method (ALM) is used. At every outer iteration if the convergence tolerance is not met the penalty for the constraints are updated together with the Lagrange multipliers if ALM is used, and the inner iterative procedure begins again.

In summary PANOC uses the same oracle as FBS methods and the operations that are the most computationally expensive operations are the prox-operator and the quasi-newton step using L-BFGS,

which still operates at high speed. These fast operations still allows the PANOC algorithm to converge superlinearly. [5]

2.5 Summary of theory

Three optimizers will be tested in an MPC application which is a control method where the input is calculated with the help of an optimizer. A whole series or so called prediction horizon of inputs can be calculated which could optimally drive the state of the system towards the reference.

Interior point method (IP) works on the interior on the feasible region with the help of barrier functions. It is allegedly good with many constraint variables and possibly bad at converging close to constraint. This is an old very common and power full method for nonlinear optimization. Another also old, powerful and common optimizer is the sequential quadratic programming method (SQP). It is an active set method and works by making second order approximations with linearized constraints of the original problem. The last optimizer Proximal Averaged Newton-type method for Optimal Control (PANOC)

3 Model Predictive Control

What follows in this chapter is the MPC formulation for this project, which will be introduced and extensively explained. Firstly the full MPC formulation is presented, what follows is the modelling of the vehicle and the environment.

3.1 Full MPC formulation for a differential drive robot

The formulation considers a *differential drive* type robot which is controlled using the inputs linear velocity, v, and angular velocity, ω, to manipulate the x and y positions as well as the heading θ. Consider the finalized MPC formulation

$$\min_{u_0,u_1,\ldots,u_{N-1}} \sum_{n=0}^{N-1} \left(\tilde{q}_n^T Q_S \tilde{q}_n + {u_n}^T R u_n + cost_{border}(q_n) + cost_{lane}(q_n) + cost_{jerk}(\Delta u_n)\right) + \tilde{q}_N^T Q_T \tilde{q}_N \quad (25a)$$

$$\text{s.t.} \quad q_{k+1} = f_{model}(q_k, u_k) \quad (25b)$$

$$d_{cone}^2 \geq ||\overrightarrow{r}||^2 - \frac{(\overrightarrow{r}\overrightarrow{V}_{1,2})^2}{||\overrightarrow{V}_{1,2}||^2} \quad (25c)$$

$$\Delta u_{max} \geq |u_k - u_{k+1}| \quad (25d)$$

$$v_{min} \leq v \leq v_{max} \quad (25e)$$

$$\omega_{min} \leq \omega \leq \omega_{max} \quad (25f)$$

where

$$q = \begin{bmatrix} x & y & \theta \end{bmatrix}^T \quad (26)$$

is the state vector and

$$u = \begin{bmatrix} v & \omega \end{bmatrix}^T \quad (27)$$

is the input vector. The costs $cost_{border}(q_n)$, $cost_{lane}(q_n)$ and $cost_{jerk}(\Delta u_n)$ along with equations (25b)-(25f) will be explained in depth in the coming sections.

3.2 Model constraints

Starting of with the vehicle dynamics, which are enforced as equality constraints in (25b). As mentioned the model at hand is a differential drive model, see Figure 1, which is nonlinear. The model constraint limits the optimization problem with respect to what is feasible for the model's dynamics. The differential drive model is characterized by turning using the difference in speed control on the right and left wheel. One of the great benefits of using this type of model is that it allows for high maneuverability since it can turn on the spot unlike a car model. Meaning that it can rotate without having any linear velocity forwards or backwards. The continuous time model is described by (28). A visual representation can be seen in Figure 1.

$$\dot{x} = v\,cos(\theta) \quad (28a)$$

$$\dot{y} = v\,sin(\theta) \quad (28b)$$

$$\dot{\theta} = \omega \quad (28c)$$

In (28) v and ω are the linear velocity and angular velocity respectively. The vehicle states $q = (x, y, \theta)$ denotes the vehicle position and heading in the world frame, see Figure 1. The model is discretized with some time step T_s which is needed in order for it to be implementable.

$$x_{k+1} = x_k + v_k cos(\theta_k) T_s \tag{29a}$$
$$y_{k+1} = y_k + v_k sin(\theta_k) T_s \tag{29b}$$
$$\theta_{k+1} = \theta_k + \omega_k T_s \tag{29c}$$

or by using the state vetor q

$$f_{model}(q, u) = \begin{bmatrix} x_k + v_k cos(\theta_k) T_s \\ y_k + v_k sin(\theta_k) T_s \\ \theta_k + \omega_k T_s \end{bmatrix}. \tag{30}$$

This is the final model constraint implemented in (25b).

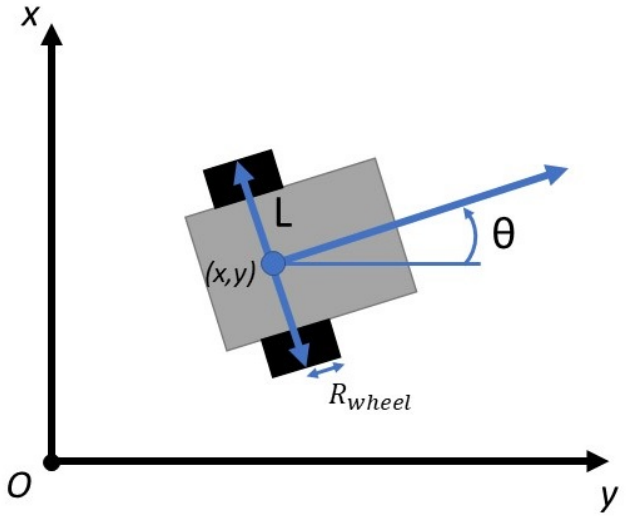

Figure 1: Description of vehicle

The vehicle will have limits on its ability to accelerate and decelerate as well as limitations on the rotational velocity, since this clearly isn't limited by the model this needs to be handled by supplying constraints. This is done by simply calculating the difference between the previously calculated velocity and the new velocity within the optimizer and setting an upper bound on the difference, $a_{max} > \Delta u = |u_k - u_{k-1}|$. This is done via (25d). The maximum and minimum velocities are simply limited to a maximum and a minimum via the inequality constraints (25e) and (25f). There is also a penalty added to limit jerk, or sudden spikes in acceleration, of the vehicle. These are penalized by the difference

$$cost_{jerk}(\Delta u_k) = R_{jerk} |\Delta u_k| \tag{31}$$

as a cost in the objective function (25a). Here R_{jerk} is a user defined scalar penalty weight.

3.3 Environmental constraints

The vehicle will be put into an environment which consists of different obstacles and rules that are defined via constraints. The environment that is chosen for this project is a two lane curved road with obstacles standing still or driving on said road. This section explains the constraints that goes into the MPC algorithm to make the model only drive within the feasible subspace. Observe the 2D environment shown in Figure 2. It consists of a number of different symbols and colors

- First, is the curved road that is a two lane road with one lane going in each direction, for the coming scenarios right hand traffic is considered.
- The ego vehicle is represented by a blue point mass with a red line that shows the heading.
- The ego vehicle also has a predicted path at each time step, this is shown as the green dots coming out of the ego vehicle, this is what has previously been called the prediction horizon.
- The green dashed line shows the path that the vehicle has driven.
- The yellow circle is the reference state which the vehicle steers towards.
- The red circle represents the obstacle and will be further explained below

The obstacles in the environment are represented by a red circle shown in Figure 3. The solid obstacle is the red filled circle which is marked with r_o in Figure 3. The dashed line in Figure 3 is $r_{obs} + r_{ego}$ which is the combined radius of the obstacle and ego vehicle, and is the radius that will be used later for the collision avoidance. The outer dotted line marked with P is the penalty margin. This idea comes from the creators of PANOC who use a similar strategy [11]. The idea is that since the optimizers will respect the constraints to a certain tolerance they will probably break them by this tolerance at some point. By introducing a penalty margin which is like an extra safety distance the optimizer might break the safety margin but the penalty for crossing the whole penalty margin radius is big enough that it wont collide with the actual obstacle. The reason why this is especially needed is how the penalties in the optimizers works. A penalty is only added if the constraints are active. This is the case if $c_{in} \leq 0$ and always true for equality constraints. In other words the optimizer would not even know about the constraint if it is not active. This means that the vehicle could drive side by side to the obstacle at the border of breaking it and only if the vehicle does drive tangent to the obstacle the penalty would be added. This is why the penalty margin must be added to make the obstacle visible before collision.

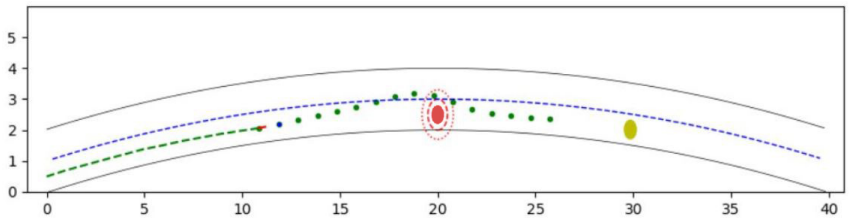

Figure 2: Curved road with ego car, one obstacle and a reference

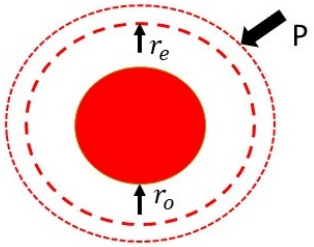

Figure 3: Obstacle

The reason for modelling the environment with curved roads is because all types of roads can be modelled with different segments of circles with different radiuses. If the radius is increased the road becomes more straight and can therefore work as a straight road segment.

3.4 Collision avoidance

The environment consist of two main types of obstacles, static and dynamic. The movement of the dynamic obstacles can be limited by a model or trajectory and thereby included in the calculations. In this project the obstacles are bound to a trajectory. This can be advantageous since the future trajectory of the obstacle can be more accurately predicted. This can represent obstacles such as humans or other undefined objects with unknown dynamics.

A interesting method for implementing collision avoidance constraint is via an old technique called collision cone. There are many different versions of this method but they build on the same concept. This method has been used for a long time to ensure collision interception in for example a missile or torpedo intercepting an aircraft or ship [18]. Since collision avoidance and interception are essentially the same problem collision cone is commonly used for collision avoidance applications. What follows is an explanation of collision cone and an extension to it in order to improve the performance.

3.4.1 Linear collision cone

The linear collision cone method uses the relative velocity of the vehicles to determine if they are on a collision course. It shows which subspace of relative velocities that guarantees collision. Choosing a velocity that is not within this subspace will therefore guarantee a collision free path. One of the main benefits of this method is that it guarantees collision avoidance even in between time steps. This can be guaranteed in each time step since it's constrained to velocities that in the future will not lead to a collision. In a sense it is predicting the collision points and avoids those.

The collision cone uses the trigonometry relationships between the two vehicles to constrain the system. Consider Figure 4. Firstly, the radius of the ego vehicle is projected onto the obstacle vehicle such that the combined radius is $R_{tot} = R_{ego} + R_{obs}$ and the ego car is treated as a point mass. Here $\overrightarrow{r}$ is the distance between the centers of the ego vehicle and the obstacle. $\overrightarrow{V}$ is the relative velocity direction of the two objects. The constraint says that $d_{cc} \geq R_tot$ must be satisfied in order to avoid collision. This means that the point mass ego car must drive pass the obstacle with a distance of at least R_tot. Note

that Figure 4 shows when the constraint is not satisfied.

The vector $\overrightarrow{V}$ needs to be converted into position vector. The vector $\overrightarrow{r}$ is projected down onto $\overrightarrow{V}$ resulting in the vector $\overrightarrow{a} = \frac{\overrightarrow{r}\overrightarrow{V}}{||\overrightarrow{V}||}$. This vector is tangent with the circle thus giving the velocity that is closest to the circle yet not colliding with it. Note that by using Pythagora's theorem on the vectors presented below the criterion can be written as

$$R_{tot}^2 \leq d_{cc}^2 = ||\overrightarrow{r}||^2 - \frac{(\overrightarrow{r}\overrightarrow{V})^2}{||\overrightarrow{V}||^2} \tag{32}$$

$$\tag{33}$$

which leads to the final constraint formulation as

$$c_{cone} = R_{tot}^2 - ||\overrightarrow{r}||^2 + \frac{(\overrightarrow{r}\overrightarrow{V})^2}{||\overrightarrow{V}||^2} \leq 0 \tag{34}$$

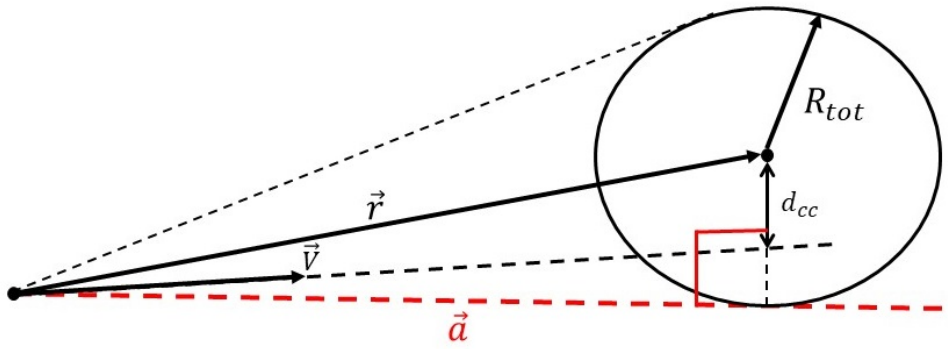

Figure 4: Collision cone illustration

As discussed earlier the cone constraints limits the subspace of allowed velocities and will therefore limit how the ego car can drive to not collide with the obstacles. There is no point in having the cone constraint active on objects that are behind the ego vehicle since it will limit how the ego vehicle can drive. The same reasoning holds for when the obstacles are in front of but very far away. If they are far away there is no need to try to avoid the obstacles and it is also unrealistic that the obstacles positions are known far away. Therefore the cone constraint is only active for obstacles that are in front and close to the ego car.

3.4.2 Non-linear collision cone

The collision cone constraint presented above is a common used method for collision avoidance for linear or close to linear systems since it predicts implicitly the collision points or the not allowed subset of velocities based on the assumption that the velocity of the obstacle is constant. If the velocity of the

obstacle changes during the prediction horizon the non-allowed subspace of velocities may be inaccurate. For example, this method does not work when the cars are moving on a curved road since the cone constraint will predict that the intersection point is off. In Figure 5 the upper arrow shows the wrong intersection point from the cone constraint given that the obstacle and ego vehicle has the same velocity magnitude.

Though the collision cone concept origins from the 60s, no nonlinear version of it was found. The proposed nonlinear version is made by the authors of this paper. The point on the road when the vehicle and the obstacle will intersect is calculated and this is the point which the vehicle must avoid. This point is shown in Figure 5 by the lower arrow.

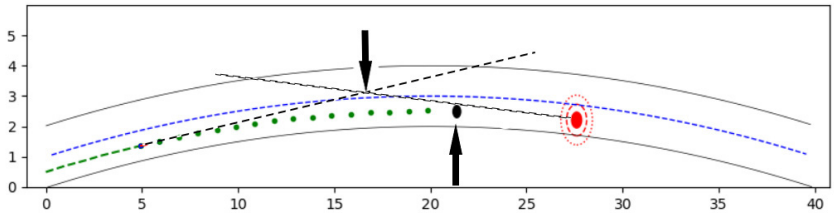

Figure 5: Intersection point (black). Ego vehicle on left with (green) prediction horizon and obstacle (red) on right.

Figure 6 shows a road segment and the imagined circle it makes if it would continue beyond its borders. It shows all the components that is needed for the non-linear collision cone to work which are described below.

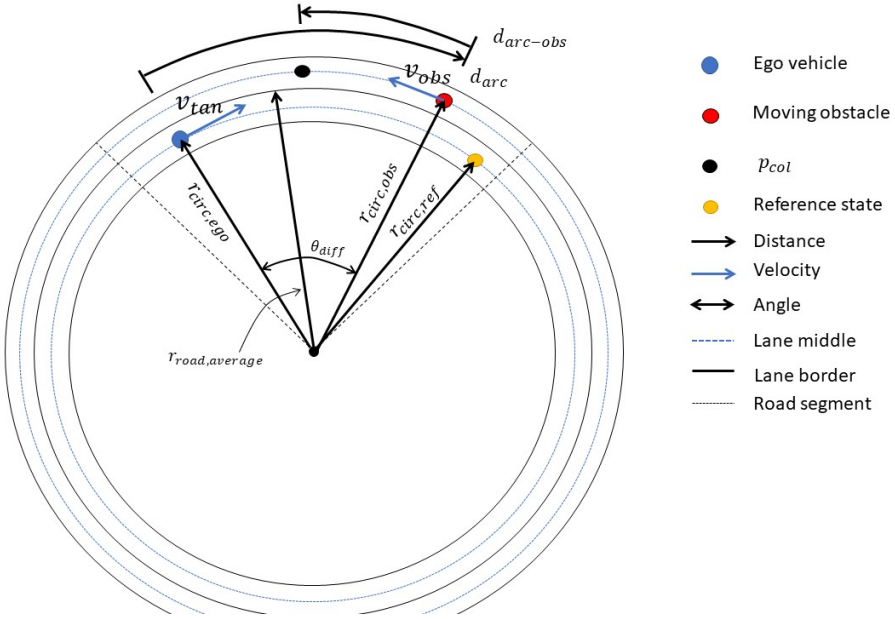

Figure 6: Nonlinear collision cone variable explanation

Firstly the ego vehicle's velocity v_{tan} that is tangent to the circle described by the curvature of the road is calculated, see Figure 6. The obstacles velocity v_{obs} is estimated given that it remains in the current lane which is one of the limitations previously made in this paper. The collision point can be calculated from the curved distance d_{arc} between the moving bodies.

$$d_{arc} = r_{road,average}\theta_{diff} \tag{35}$$

$$t_{impact} = \frac{d_{arc}}{v_{tan} + v_{obs}} \tag{36}$$

$$d_{arc-obs} = d_{arc} - d_{obs} = t_{impact}v_{obs} \tag{37}$$

Here $r_{road,average}$ is the average radius from the ego car and obstacle to the circles centre. The collision point p_{col} is calculated from $d_{arc-obs}$, the distance the obstacle must travel during the time until impact t_{impact}.

When this collision point described by the black dot in Figure 5 shown by the lower arrow, the collision cone constraint is applied to a fictive obstacle that is moved to that position. The velocity of the obstacle is also set to zero. This way the collision cone can effectively be used on nonlinear environments. In short it calculates the point were they will collide and then the linear collison cone is used on a fictive obstacle moved to that point with 0 velocity.

This method has one flaw that has not been addressed in this project. Since v_{tan} will change during the prediction horizon in each prediction, meaning a pair (y_k, x_k), v_{tank} can change which in turn changes p_{colk}. So in each prediction the car will try to avoid an obstacle that is slightly shifted forwards or backwards on the road from what the previous prediction of p_{col} if the car changes v_{tan}. In the simulations however the intersection point is only moved slightly and the method has been proven in these simulations to be quite effective. This is because the point is only moved slightly between each prediction since the car cannot turn and accelerate very fast and each prediction point (x_n, y_n) has an accurate intersection point $p_{col,n}$. A possible improvement would be if the collision point could be estimated given the whole prediction horizon instead of doing it at every prediction step. With this new method the collision cone described in 34 can be applied and this is what represents the collision avoidance constraint in the MPC formulation (25c).

3.5 Border constraint

To ensure that the vehicle drives within the road, two border constraints are introduced. They act as repulsive regions if the vehicle drives to close to the edge of the road. These constraints are on the form, $(ad_{center})^3 + bd$ where d_{center} is the distance from the vehicle to the centre of the lane next to the border and b is a penalty factor. An example of such a function is shown in Figure 7. One must scale d_{center} such that the function takes on appropriately high values at a given distance d_{center} from the centre of the lane. This is done by the scaling factor a. The cost that is added to the objective function (25a) becomes

$$cost_{border} = max(0, (ad_{center})^3 + bd_{center}) \tag{38}$$

where max ensures that only the positive part of the function is accounted for. This method allows some unwanted behaviour, since it's allowed for the vehicle to leave the road. Just with an increased cost to

the planned path. This case is most evident when the road is completely blocked by obstacles and some optimizers still plans a path around the obstacles outside of the road.

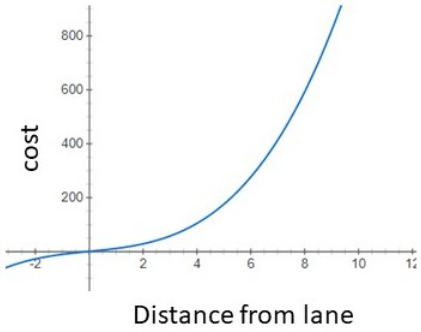

Figure 7: Border constraint

A reason for not having hard constraints on the border is related to ethics and will be discussed in Section 9.7. Another benefit of having soft border constraints stems from the border then becoming repulsive due to the increased cost. This pushes the ego vehicle to stay away from the borders and simplifies the overtaking maneuvers by eliminating local minimums that occur between the obstacles and the road border. These minimums are shown in Figure 9 by the blue dots. Thus it was chosen to keep the borders as soft constraints.

3.6 Moving the reference state and lane keeping

The optimizers will decrease the objective function by minimizing the distance between the vehicle and the reference point. The fastest way to minimize this distance is to go in a straight line towards it as shown in Figure 8 by the lower arrow taking the birds way on this circular road.

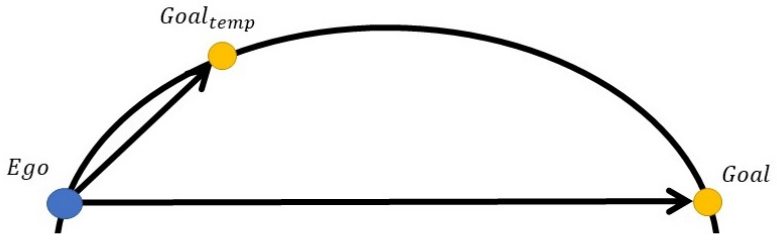

Figure 8: Fastest path

This introduces a problem when the road is curved as the car wants to drive off the road in a straight line towards the goal. To make it go in the curvature of the road a soft constraint is introduced. This

constraint adds a penalty to straying far away from the centre of the lane which it shall drive on, which is the lane which the reference point is placed on. By doing this one can increase the likelihood that is stays on the road. The cost is defined as

$$cost_{lane} = b_{lane}\sqrt{(r_{circ,ego} - r_{circ,ref})^2} \tag{39}$$

and is the final piece of the MPC formulation in 25a. Where $r_{circ,ego}$ is defined as the radius from the center of the circle that represents the road to the state where the ego vehicle currently is positioned. See Figure 6. The constant $r_{circ,ref}$ is defined as the radius from the center of the circle that represents the road to the state where the reference is positioned. Here b_{lane} is an user defined weight which determines how hard the ego vehicle is penalized for straying away from the reference lane. However this constraint is only active when the cone constraint is inactivated. This is so that it does not limit and interfere with the collision avoidance system which is of higher priority.

The bigger the curvature the more apparent the problem of staying on the road becomes and this soft constraint might not be enough. The core of the problem is that the curvature between the reference and the car is not included enough in the objective function so that it is not bended with the curvature of the road or that no constraints tells the optimizer that these points are infeasible. One method of dealing with this is by moving the reference point along the lane closer towards the car as can be seen in the Figure 8 by the $Goal_{temp}$ instead of the original $Goal$. This makes it easier for the algorithm to optimize towards the reference point. This is also a realistic approach since the car does not necessarily have information about the road far away. The new temporary reference is moved to a point on the same lane as the original reference point at a distance $v_{max}NT_s + d_{ref,margin}$ away. Here v_{max} is the ego vehicle's max velocity, N is the prediction horizon, T_s is the time step and $d_{ref,margin}$ is a chosen short distance ensuring that the last prediction step doesn't fully reach the reference point.

3.7 Terminal cost and reference angle

Ideally one wants to give the vehicle much freedom in planning the path in between the current position and the reference position to not limit the optimizers ability to comply to the constraints. One way of manipulating this is by emphasizing the terminal cost in comparison to the stage cost in (25a). By lowering the stage cost and increasing the terminal cost, Q_T, the vehicle can plan each step in the prediction horizon more freely while still closing in on the reference point.

To reduce the amount of local minimas in the objective function one could make a simplification in (25a). That simplification is to set the element in Q_S and Q_T that penalizes the angle error $\tilde{\theta}$ to 0. In other words $Q_S(3,3) = Q_T(3,3) = 0$ for all $\tilde{\theta}$. This implies that the optimizer doesn't care about minimizing the angle, thus making it less prone to follow unwanted local minimas. Further, if necessary, due to how the vehicle is designed it can easily compensate for this angle once it has reached the end position (x_r, y_r)

One flaw with this approach is that in order to be able to accurately calculate the collision point the velocities of the obstacle must be somewhat predictable. In this thesis the obstacles moves on only one lane and with constant speeds.

3.8 Summary Model Predictive Control

The scenarios the ego vehicle will drive in is modeled using soft and hard constraints. The car has a few restrictions and characteristics which has been modeled so that the simulated car will behave similarly as the Arduino robot. The circular road segment that the optimizers will be tested in has been modeled with two lanes and constraints in order to keep the car within the road borders. The anti collision system is implemented using the collision cone method with a proposed nonlinear version of it to be able to use it in nonlinear environments.

4 Processing inputs and outputs

To get a better result some precautions had to be done to the inputs and outputs. Firstly, supplying good initial guesses for the optimizer by using warm starting will be discussed. Warm starting potentially allows for faster convergence to an optimal trajectory, but also makes the algorithm more prone to converge to some local minimas. Secondly, it's discussed how to handle paths supplied by the optimizer where it hasn't converged to an acceptable set of states.

4.1 Warm start

A common feature in optimizers is the ability to warm start them. This means that the optimizer is supplied with an initial value based on previous actions. This can be a very effective method that speeds up the optimizer if the environment does not contain many local minimas. If the environment contains local minimas such as the blue dots in Figure 9 the optimizer is in some cases more likely to end up in those if warm start was used. This depends on how the problem is defined and where the minimas are located as well as the optimizer that is used. Figure 9 shows PANOC when operated with warm start and as one can see when it avoided the first obstacle it was already moving upwards so that it got stuck in the local minima of driving above the second obstacle. This is because the optimizer was fed a solution that was close to a local minima. So if the optimizer is initialised close to a local minima it's prone to get stuck in that minima than converge globally. In these cases it is sometimes much better to cold start the optimizer so that it is more likely to take different actions. As mentioned earlier the solution could converge faster if warm started but it could lead to an overall slower convergence since as the case in Figure 9 it will eventually end up in an area where the border cost is really high and this slows down the convergence substantially.

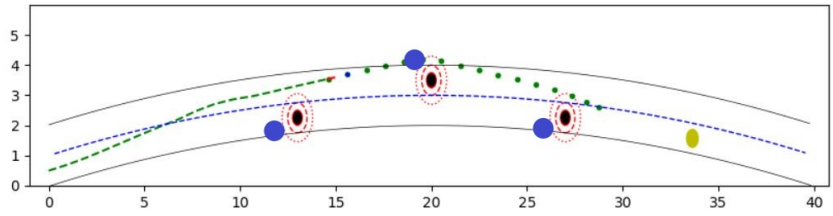

Figure 9: PANOC warm start

4.2 Bad Paths

It is a common in MPC optimization to turn hard constraints into soft ones by including them in the objective function in terms of penalties. This is especially common when testing on real life applications as disturbances can more easily end up making the problem infeasible. PANOC is defined to only have hard constraints on the input variables and soft constraints for everything else by relaxing these constraints by using the augmented Lagrangian method or the penalty method. SQP and IP also relax their constraints for the same reason. The constraint relaxation methods comes with a few cons. They can lead to a solution where the optimizer chooses an invalid path due to the penalty of violating the constraints still gives a better next iterate than not violating them if the penalty is to low. On the other hand, choosing a high penalty can cause the problem to become ill conditioned, which makes it much

harder for the optimizer to converge. Thus, the problem becomes very dependent on choosing the right penalty. Too high and it will not be able to converge at all and too low it will not respect the constraints.

The optimizers have a number of defined convergence criterias which tells it that the solution is good enough. Such criterias can be reaching the minimum step size, no decrease in objective function or the constraint tolerances has been met. The optimizers can however stop prematurely due to that it runs out of function evaluations, iterations or time. There is then a risk that it has not converged. The optimizers will then output the best iterate it has found. It is a common feature in optimizers to output a solution no matter if a convergence criteria has been met or not. It then says if it has converged or not. However, to simply check if the optimizer has converged or not is not enough, since the outputted paths still could be unacceptable. Therefore the feasibility of the solution given has to be checked and handled outside of the optimizer. One solution to this is to simply check the suggested path for violations and if the solution is deemed to be invalid the ego vehicle stops and begins a new search. However, in many cases stopping instantaneously is unrealistic or unwanted. Instead the most recent acceptable path is saved and if the new path is deemed unacceptable the car continues to drive on the last accepted path.

To determine if the found solution is valid the calculated path is fitted to a forth degree polynomial of the form $ax^4 + bx^3 + cx^2 + dx + e$. If the error between the fitted polynomial and the output path is above a certain threshold the path is deemed bad and the computed control input will be ignored and the input from the last accepted path is used until the optimizer can supply a new path with an acceptable fitting error. This threshold needs to be decided from experiments. Figure 10 shows an acceptable path and Figure 11 shows an unacceptable path. The plot right below the road plot is a visual example of the curve fit function where the blue is the actual prediction horizon and the red is the fitted curve. The number in the lower right corner is the fit error between the two curves.

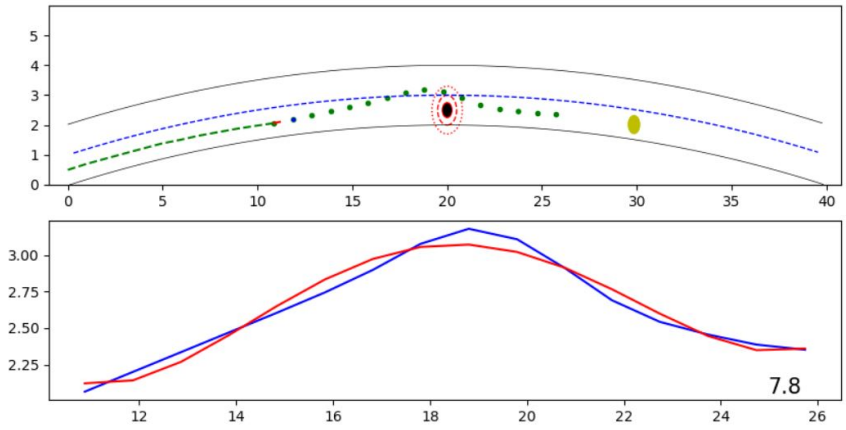

Figure 10: Acceptable path. Fitting line in red and planned path in blue.

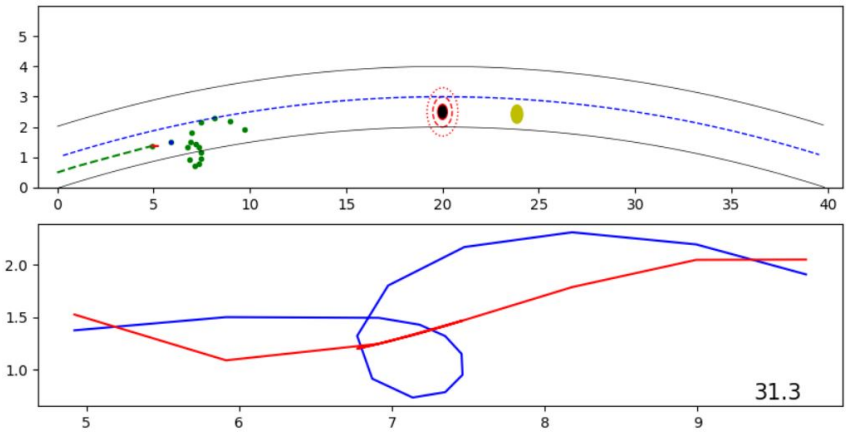

Figure 11: Non acceptable path. Fitting line in red and planned path in blue.

This is a simple way to quickly judge if a path is valid or not. However, there are still some bad paths that can pass this curve fitting test. The problem with this test lies in when lowering the fit error value the space of possible paths are restricted which is not necessarily desirable. In other words it can filter out some paths that does not fit the forth degree polynomial, but in fact are acceptable paths. Another problem is that this method does not take into account the environment, only the shape of the path. The optimizer could therefore plan a path outside the environments bounds that would be deemed valid. To tackle this problem much more logic would have to be programmed to deal with all the different cases. However, it might not be a good strategy since one is not fixing the problem at its root. Tackling the problem at its root would be to improve the optimizers convergance capabileties so that it does not output bad paths that often. Some examples of these bad paths are shown below in Figure 12. This problem will not be fixed here and will be used as a measurement of comparison between the optimizers later on. It is hard to define what is a bad path and not. Paths that one can clearly see has not converged are deemed bad.

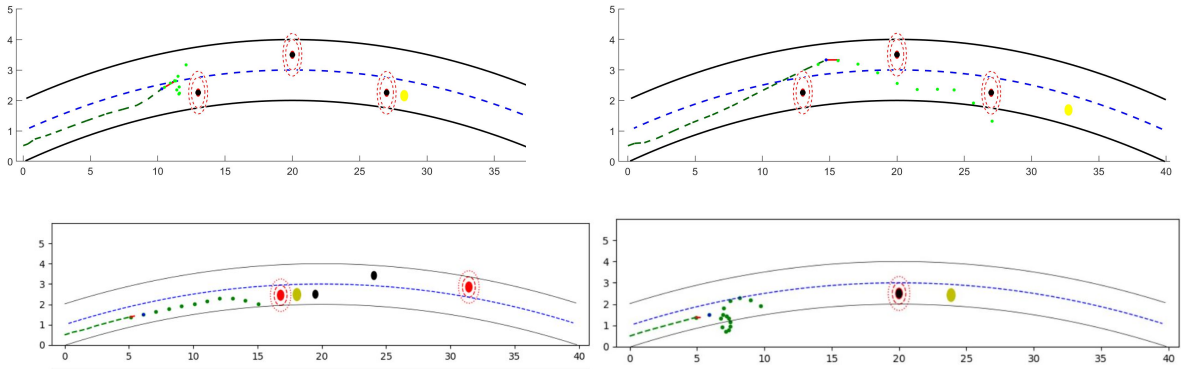

Figure 12: Bad paths

4.3 Summary processing inputs and outputs

To potentially ease the job of the optimizers the concept of warm start is introduced. This is when the optimizer is initialized with a solution close to the optimal solution. The concept of bad paths has been brought up which essentially is a path that has not converged and could possibly lead to applying the wrong input signals to the car. The output from the optimizers must be checked so that it meets the requirements which is primarily done by a curve fit of the calculated path onto a forth degree polynomial.

5 Evaluation framework

In this section the scenarios in which the algorithms will be evaluated will be presented. Further, it will also introduce and discuss the methods and measurements with which the algorithms will be evaluated.

5.1 Scenarios

Three scenarios were designed to evaluate the algorithms. The aim is to design each case to specifically test how the algorithms handles environments where everything is static, environments with gradual change and environments with sudden change. All the roads are curved with a centre radius of 100 units. The radius of the ego car and obstacles are 0.125 units and the lane thickness is 1 units. Each scenario is simulated with a horizon of 10, time step of 0.5, $0.1 < v < 3$, $-3 < \omega < 3$, $dv < 0.5$ and $\omega < 0.5$. SQP and IP are warm started whilst PANOC is cold started. The distance where the car can see obstacles is 5 units. Each scenario is presented in greater detail below.

Overtaking maneuver of three static obstacles

Firstly, the algorithms are evaluated on a simple slalom track, see Figure 13. The ego vehicle starts on the bottom lane. This track have multiple local minimas, which must be avoided in order for the algorithm not to get stuck. Two minimas are found in between the two lower obstacles and the lower road border and one is found in between the upper obstacle and the upper road border. These minimas are also shown in Figure 9. It's also very constraint intensive, since for the most part of the simulation at least one obstacle will close enough so that a collision cone constraint is active.

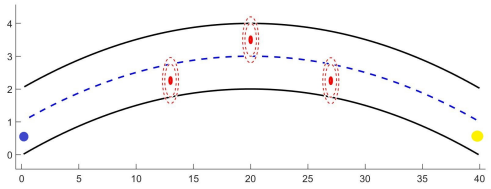

Figure 13: Scenario with three static obstacles

Overtaking maneuver of two moving obstacles

To observe how to the ego vehicle is handling dynamic environments an overtaking scenario was constructed. One obstacle is moving along the lower lane towards the right, while another obstacle is moving towards the left along the top lane, see Figure 14. The obstacles are constrained to move only on the middle of the lane where they were initiated on and this is included in the optimizer. In other words the optimizers know the trajectories of the obstacles in beforehand. The ego vehicle will start on the bottom lane, overtaking the obstacle in front of it and return to the bottom lane.

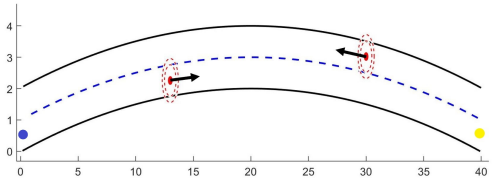

Figure 14: Scenario with two moving obstacles

Suddenly appearing obstacle

In this scenario the aim is to be able to evaluate the algorithms ability to handle sudden changes in the environment. The obstacle will appear on the position shown in Figure 15 when the obstacle is some predefined distance away.

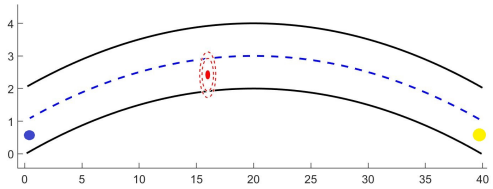

Figure 15: Scenario with one suddenly appearing obstacles

5.2 Evaluation methods

To somewhat compare the optimizers two measures of path quality was defined, *number of bad paths* and *robustness*. The computation time of each horizon was also recorded and plotted. It is hard to define what a bad path is which was discussed in Section 4.2, however this will serve as pointer in the general direction of which optimizer that tends to converge to better paths. *Robustness* is measured by simulating 10 consecutive times with added normal white Gaussian noise with mean 0 on both v and ω. The standard deviation σ is kept as big as possible and if the car can drive without hitting any obstacle or go outside the road with 80% success is deemed successful. The maximum standard deviation that could be added is shown under the column *Robustness*. The angular velocity ω is much more sensitive to noise as it could align the car in an extreme angle towards an obstacle or the border of the road making it harder to compute a feasible path. The noise on the velocity is much less likely to affect the robustness. Therefore when choosing the standard deviation the following start values were chosen $\sigma_v = \frac{1}{2}$ and $\sigma_\omega = \frac{1}{100}$. The values were decreased gradually until success. The values were then increased separately to find the highest possible value for both of the input. This test was not performed in such quantity so that it is statistically accurate. The purpose is mainly to get an idea of the robustness so that the correct algorithm can be chosen for installation in the hardware.

5.3 Summary evaluation framework

Three scenarios was introduced in which the car will drive. They all pose different challenges. The scenarios are; overtaking maneuver of three static obstacles, overtaking maneuver of two moving obstacles

and suddenly appearing obstacle. A few tests and things that is going to be documented based on the scenarios is brought up such as a robustness test and comparison of computation time.

6 Simulation results

The scenarios are simulated with PANOC and MATLAB's SQP and IP algoritms. Under each scenario plots from three different time steps for each algorithm is presented. Next to each set of time step plots there is also a plot of the corresponding computation time for each prediction horizon. Note that the computation times presented in the figures are in milliseconds for PANOC and in seconds for SQP and IP. Each scenario ends with a table comparing some data from each optimiser.

Overtaking maneuver of three static obstacles

All algorithms were able to complete this scenario and the results are shown in Figures 16, 18 and 20. In Figures 17, 19 and 21 it is clear that PANOC is superior when it comes to computation speed. Further, it's clear that PANOC works well enough and can plan an many acceptable trajectory. SQP doesn't struggle either and is the only algorithm that does converge to sensible paths throughout the simulation, see Table 1 *bad paths*. IP plans the least optimal trajectory of all the algorithms, which can be seen in Figure 20 where the planed trajectory is quite wiggly and plans longer paths around the obstacles. Little can be said about the robustness from Table 1, other than that PANOC seems to handle disturbances better that both of the MATLAB algorithms.

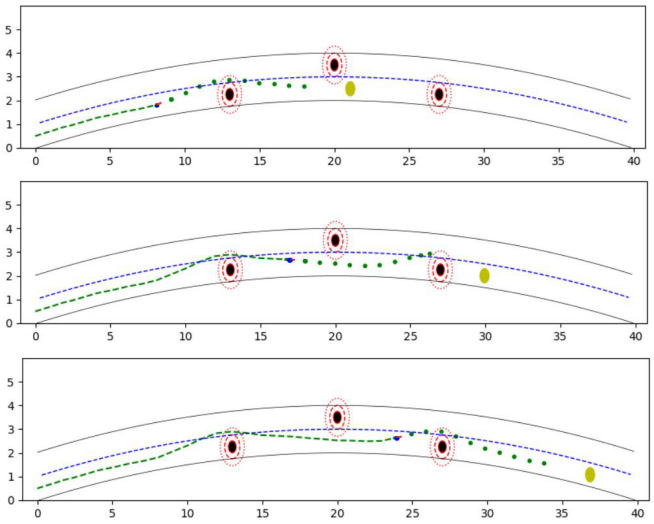

Figure 16: PANOC overtaking three obstacles

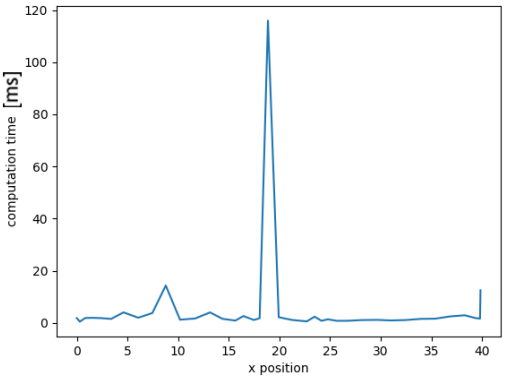

Figure 17: PANOC optimizer computation time

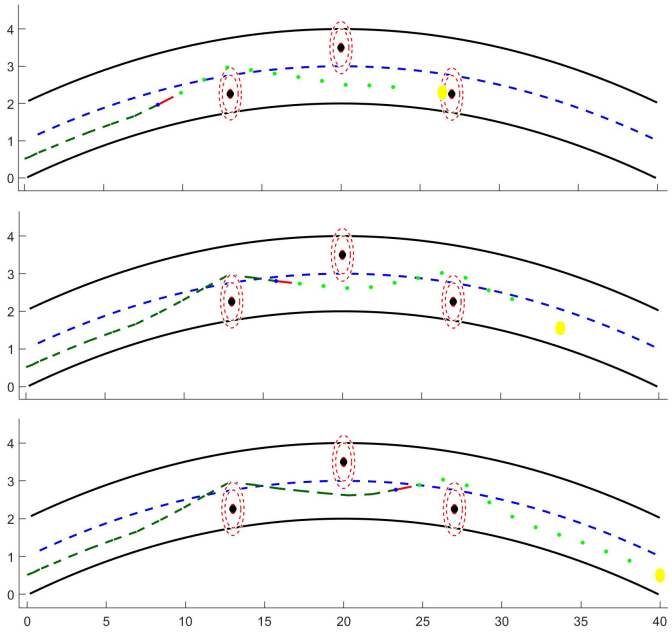

Figure 18: SQP overtaking three obstacles

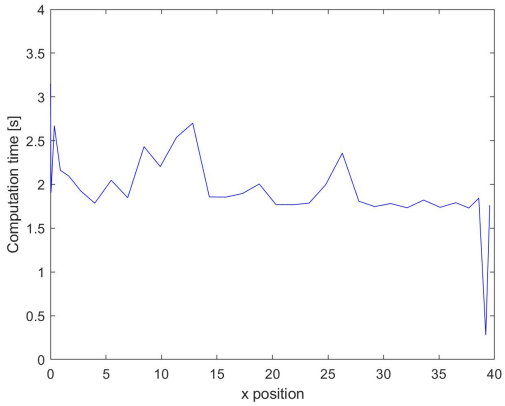

Figure 19: SQP computation time

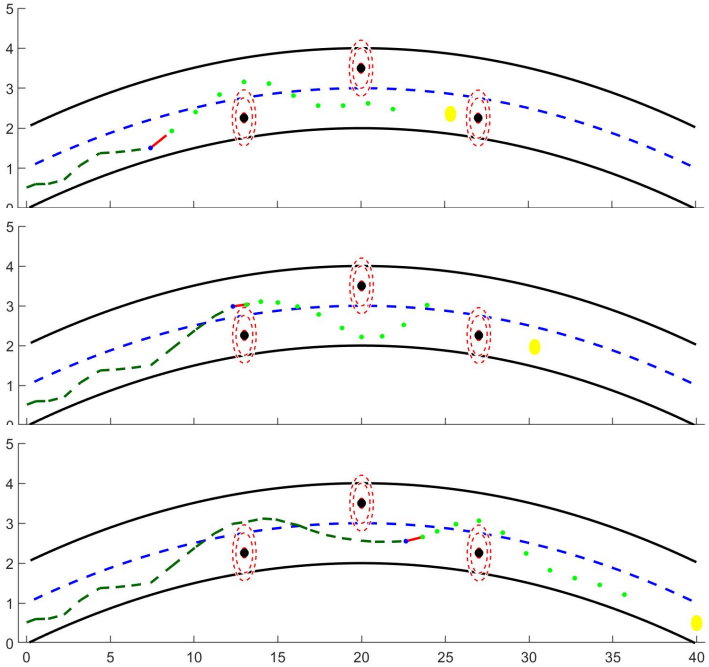

Figure 20: IP overtaking three obstacles

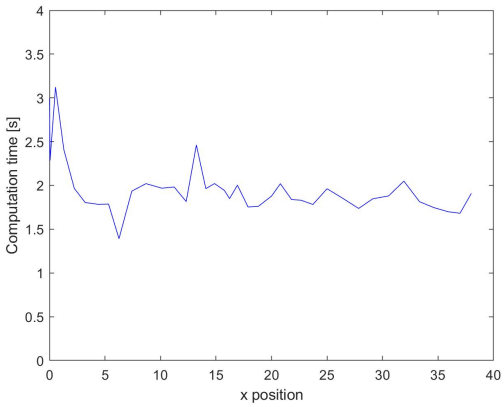

Figure 21: IP computation time

Three static obstacles		
	Bad paths	Robustness
PANOC	ca 8	$v_d = 1$, $\omega_d = \frac{1}{600}$
SQP	0	$v_d = 0.1$, $\omega_d = \frac{1}{1500}$
IP	3	$v_d = 0.1$, $\omega_d = \frac{1}{1500}$

Table 1: Simulation results three static obstacels, comparing SQP, IP and PANOC

Overtaking maneuver of two moving obstacles

All algorithms were successful in finding a path around the obstacles in this scenario. The results are presented in Figures 22, 24 and 26. From Figures 23, 25 and 27 it's still clear that PANOC outperforms the other two algorithms with respect to computation time. PANOC plans a quite bad path with a lot of overshoot at the first obstacle. It has clearly not planned an optimal path which also can be seen in time step one and two in Figure 22 where the prediction horizon has not converged in the end. SQP might have found a more optimal path, however it has some problems planning a smooth path around the $x = 6$ mark. IP is still having problems with the planned trajectory converging to non-sensible paths, but is still able to pass the two obstacles. From Table 2 SQP still only outputs sensible paths, while PANOC and IP have a few bad paths. Regarding robustness the situation hasn't changed from the scenario with three static obstacles. The take away from this scenario is that it's not obvious if PANOC or SQP is the better optimizer. SQP is more optimal but slower, while PANOC is faster but less optimal. PANOC is also more robust, see table 2. Further IP is still struggling and does not match up to the other two algorithms.

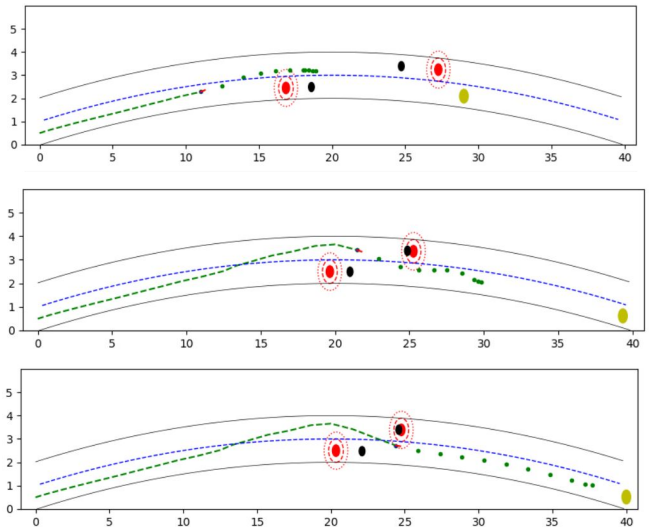

Figure 22: PANOC overtaking of two moving obstacles

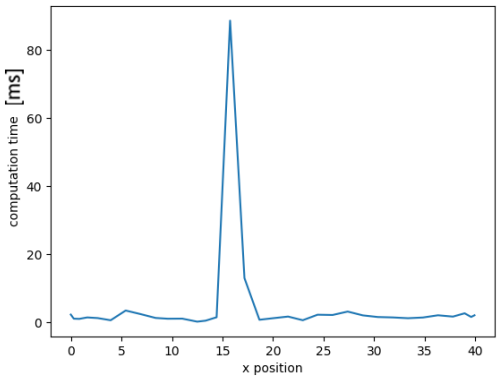

Figure 23: PANOC optimizer computation time

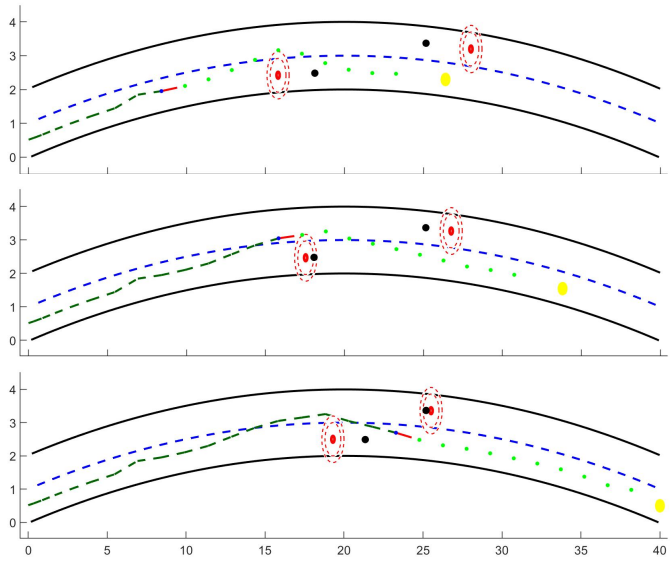

Figure 24: SQP Overtaking of two moving obstacles

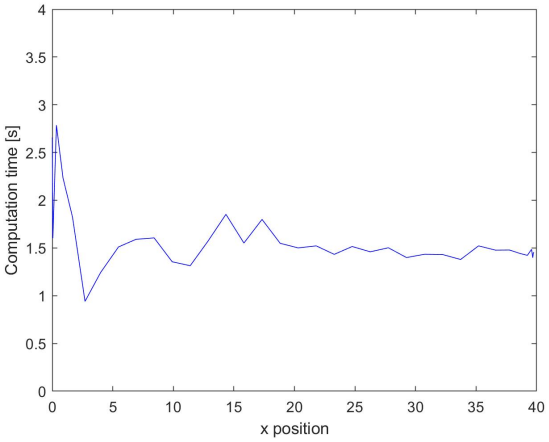

Figure 25: SQP computation time

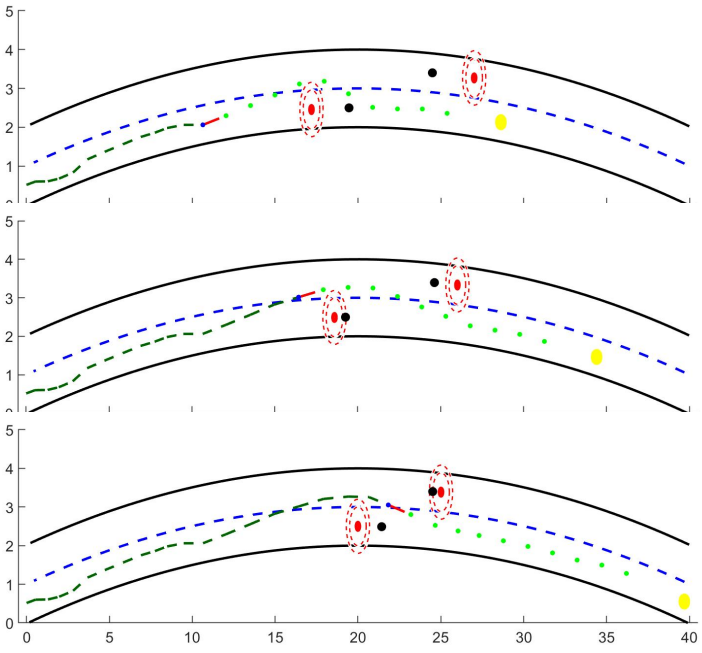

Figure 26: IP Overtaking of two moving obstacles

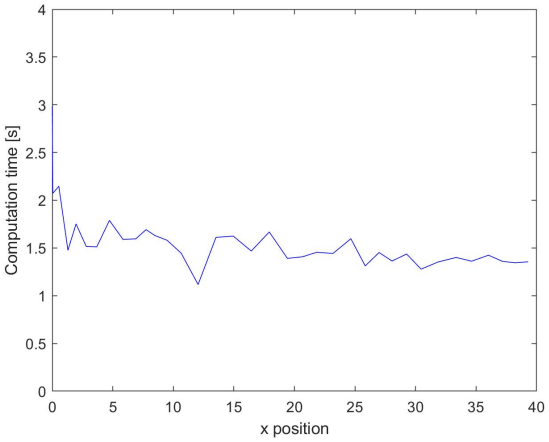

Figure 27: IP computation time

Two moving obstacles		
	Bad paths	Robustness
PANOC	ca 6	$v_d = 1$, $\omega_d = \frac{1}{600}$
SQP	0	$v_d = 0.1$, $\omega_d = \frac{1}{1500}$
IP	3	$v_d = 0.1$, $\omega_d = \frac{1}{1000}$

Table 2: Simulation results two moving obstacles, comparing SQP, IP and PANOC

Suddenly appearing obstacle

All algorithms can handle this scenario, as can be seen in Figures 28, 30 and 32. However, in this scenario there was a clear hierarchy to which algorithm that performed best. The reason for this is the evaluation method *Activation distance* presented in Table 3. This distance is the distance between the car and the obstacle, which activates the suddenly appearing obstacle when the car is at this distance away from the obstacle. Here it can be seen that PANOC is best at handling suddenly appearing obstacles, since the obstacle can appear closer to the ego vehicle. SQP is second best and last is IP. In Figures 29, 31 and 29 PANOC is the fastest algorithm. From table 3 PANOC is the most robust, while SQP is best at avoiding bad paths. The take away from these results is that PANOC is best at converging in tight situations. It could be because it has a greater ability to converge from an arbitrary point. Remember that the other two algorithms are warm started and when the obstacle appear this warm started value is quite inaccurate.

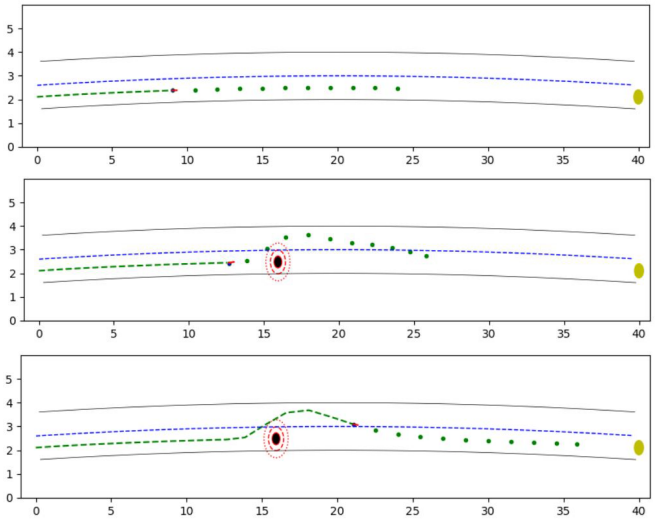

Figure 28: PANOC avoiding a sudden obstacle

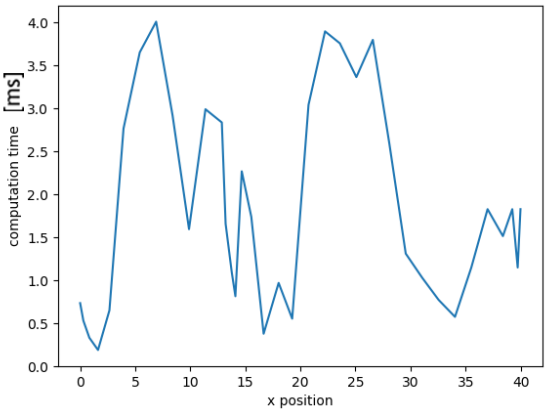

Figure 29: PANOC optimizer computation time

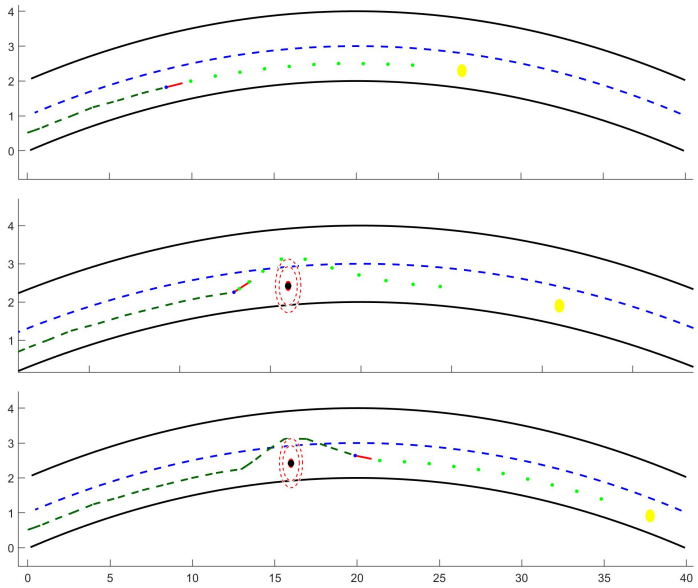

Figure 30: SQP avoiding a sudden obstacle

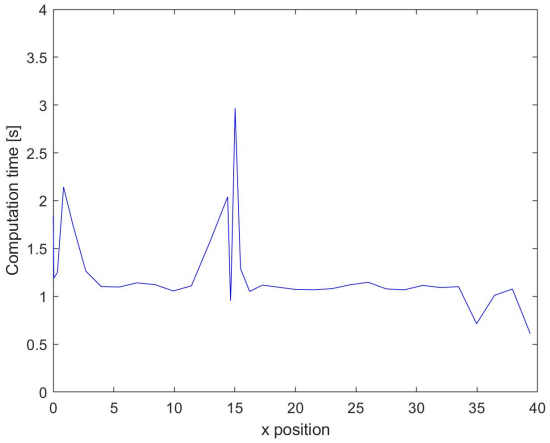

Figure 31: SQP computation time

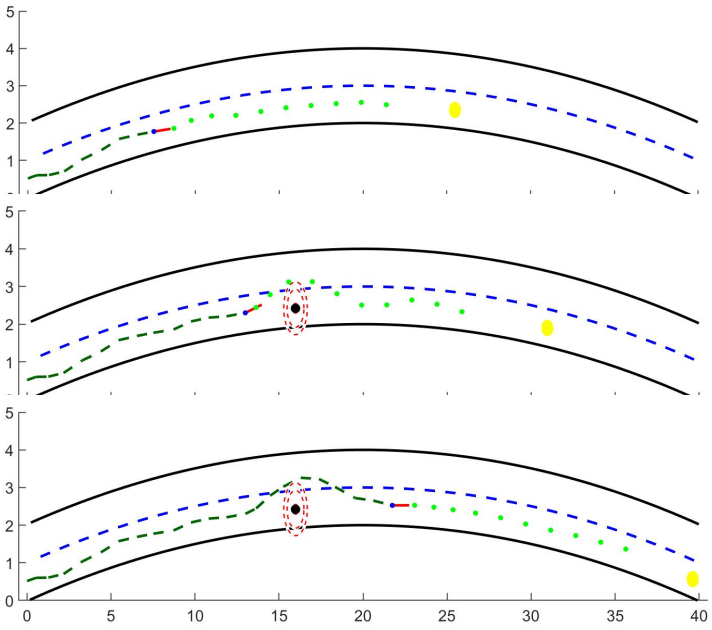

Figure 32: IP avoiding a sudden obstacle

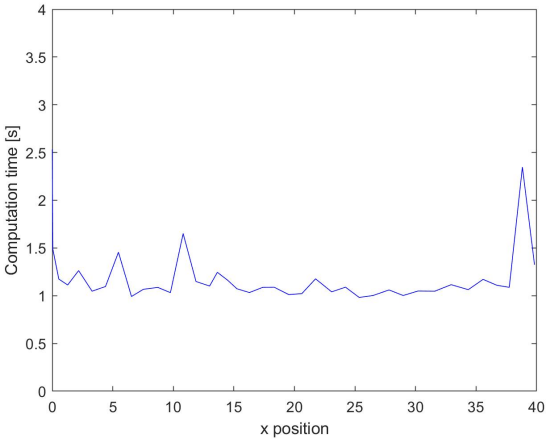

Figure 33: IP computation time

A suddenly appearing obstacle			
	Activation distance	Bad paths	Robustness
PANOC	4.5	ca 7	$v_d = 1$, $\omega_d = \frac{1}{500}$
SQP	5.5	0	$v_d = 0.1$, $\omega_d = \frac{1}{1500}$
IP	7	2	$v_d = 0.1$, $\omega_d = \frac{1}{1000}$

Table 3: Simulation results suddenly appearing obstacle, comparing SQP, IP and PANOC

6.1 General observations

One thing that is not presented in the results above is that the IP algorithm take smaller steps distributed over the whole prediction horizon. In other words it does not drive with the maximum allowed velocity. This means that, in the simulation, the algorithm takes smaller steps than the other algorithms towards the reference state. Also, the optimizer indicate that it has not converged fully, further validating the claim. This is especially noticeable when it is approaching the reference state. It has a hard time converging when the solution closes in on a constraints. Further, the IP algorithm doesn't converge in the first two simulation steps, telling us that without warm starting the algorithm has a harder time converging with the given parameters.

For SQP there were some changes in hyper parameters needed, in order for the algorithm to clear the track the constraint tolerance had to be increased, otherwise it risked getting stuck when trying to pass the obstacles in local minimas. Specifically those minimas presented in Figure 9. However, this this modification still allowed the system to behave well.

All the algorithms needed a lot of tuning. To get a nice smooth path planning in the majority of the iterations one had to tune for some time. IP and SQP had the tendency to get stuck whilst PANOC always made it to the end. However, during some sessions PANOC did go off the road when the present situation proved too difficult and it wasn't tuned correctly. Further, PANOC also planned more suboptimal paths. It often produced acceptable paths, but they were generally in the neighbourhood of the solution, but didn't fully converge into the minima. Giving the algorithm more inner iterations or more computation time did not improve the overall result.

To conclude the results presented above, it's clear that PANOC is the fastest algorithm, and also the most robust in regards of disturbances. Further, PANOC is the one where the planned path quickest adjust itself to sudden changes in the environment.

6.2 Convergence test

One of the big problems was that the optimizers did not converge to an optimal path. It was clear when running the simulations that sometimes the output from the optimizers had not converged. When iterating the inner procedure the optimizers needs to ask the oracle to get information about the main minimization function. These are for example derivatives and function values for different parts of it. In SQP and IP the algorithms needed much more than the default number of function evaluations to converge so a test was set up to let the optimizers converge fully and the result was recorded in the figures below with both warm and cold start. All of the following results are recorded from the scenario with two

moving obstacles presented in Section 6. The number of function evaluations is a good way to compare two otherwise very different algorithms. By limiting the number of function evaluations one is essentially limiting the information it can access. As can be seen in Figures 34 and 35 below the correlation between the number of function evaluations and the computation time is strong.

The effect on not using warm start is dramatic in both the SQP and IP algorithm. When running with cold-start the optimizers are much more likely to not converge which is expected. In highly constrained areas they will not converge within a reasonable time relative to the warm start. IP will use up all the function evaluations in many cases. In these tests the upper limit for function evaluations is $10e4$ which is a lot. The other iterations where IP does not use all function evaluations it will converge to an infeasible solution. Furthermore the resulting path will also be infeasible so even with this high limit it does not manage to drive the scenario with cold-start. In this enviroment it is not globally convergent from any arbitrary starting point. SQP will get stuck in a mix of unfeasible solutions and solutions where the objective function is non-decreasing. The latter does not necessarily mean convergence to an acceptable path as shown in Figure 36 where at the end the majority of the solutions were of those kind. It still did not manage to output any acceptable path. It has somehow found a local minimizer which does not look right at all. Even though it says it has found a minimizer it is doubtable and the question arises in this case if there is some flaw in the algorithm which results in this behaviour. It is hard to tell why it behaves like this as the optimisation function is hard to visualize and it is hard to identify clear local minimas.

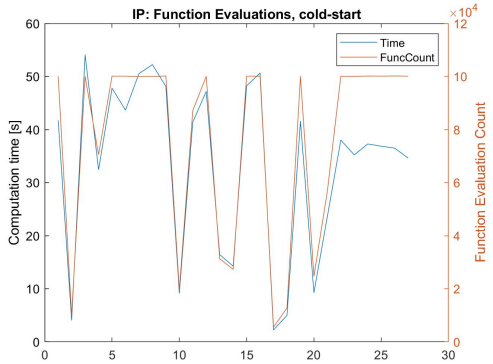

Figure 34: Cold start IP

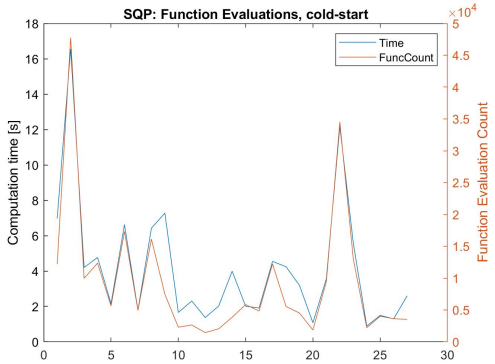

Figure 35: Cold start SQP

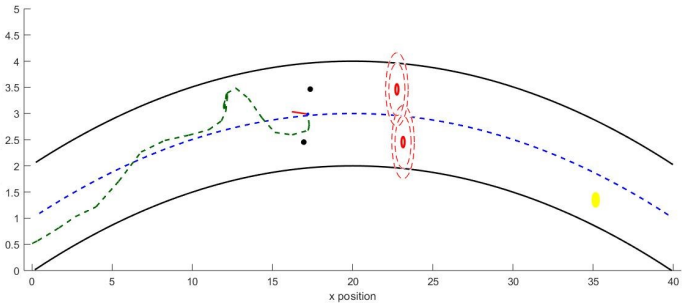

Figure 36: SQP cold start

When the optimisers are warm started they reach the goal. However one notice a big difference on the SQP and the IP algorithm in the number of used function evaluations. This is actually a characteristic difference between them where IP in many cases needs more function evaluations.[19] This was seen through the project that in many cases IP could run for longer and keep asking for function evaluations to converge before meeting the requirements. This does not necessarily say that one is faster than the other. It can though be an indicator that this problem is more suited for SQP compared to IP.

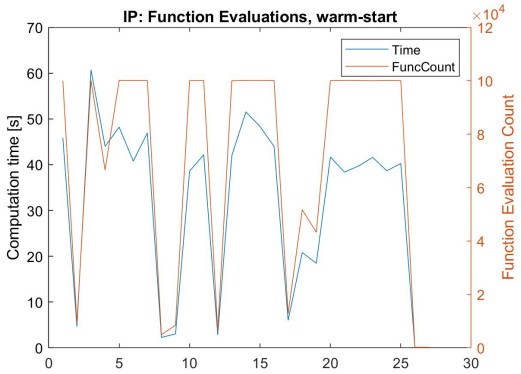

Figure 37: Warm start IP

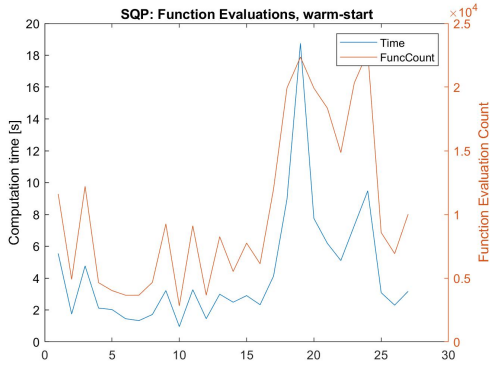

Figure 38: Warm start SQP

PANOC does not output the number of function evaluations. The next best comparison is the inner iterations. Each time the algorithm takes a step it uses up an iterative which itself uses function evaluations to determine how far this step should be. The number of inner function evaluations is not accessible. There is, however, a strong correlation between inner iterates in PANOC and the computation time. The important thing to see in Figure 39 and 40 is that PANOC does not need significantly many more iterations to converge from cold start. In terms of computation time it does not make much of a difference to cold start the algorithm. The difference with warm start is merely 85% faster when comparing the median. The operations in PANOC are simpler and faster than those in IP and SQP which both only had a median of 43 number of iterations.

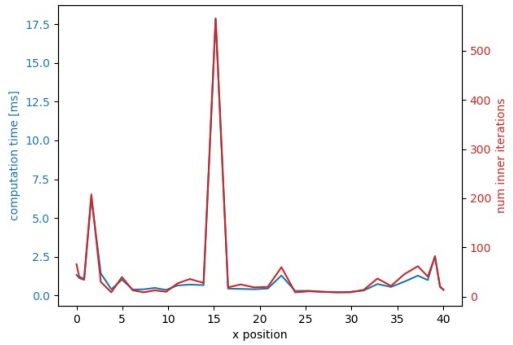

Figure 39: Warm start PANOC

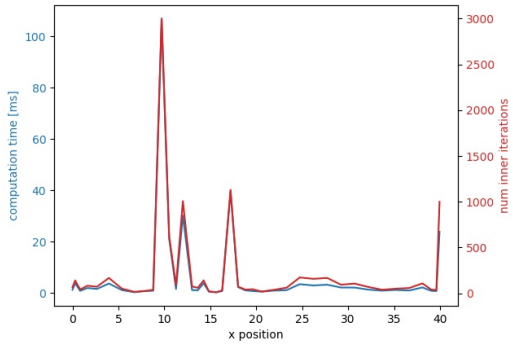

Figure 40: Cold start PANOC

Some more tests were made where PANOC had significantly more computation time, outer and inner iterative than in the test above. This did not result in any improvements which raises questions on how well PANOC can converge globally. However, it can evidently converge locally with speed and robustness.

6.3 Summary simulation results

IP did not perform well enough to be considered for implementation. It was not fast enough and did not converge close to the optimal solution even when warm started. It demands a lot of function evaluations and even when given extra time to converge it fails to do so well enough. The IP method does not seam to be well fitted for these kinds of environments or problems.

SQP performed similarly to IP in some aspects. It to was slow and had to be warm started. Though one big difference was that it could converge quite optimally with high reliability. Since it must be warm started and is slow it is not suited for implementation.

PANOC was really fast and converged quite well in the environment. It was a clear winner for the purpose of implementation. It can be cold started and is fast so the results are promising enough to be implemented in the hardware. The main problem with PANOC was that is sometimes had a non convergent behaviour and outputted more bad paths than any other algorithm. Even though it sometimes had a non convergent behaviour it was still the most robust optimizer and could drive the car to the reference most often and better than the others when subject to disturbances.

7 Implementation

PANOC is a clear winner in the evaluation of the three algorithms. It is the fastest and can better handle disturbances. Thus, PANOC will be used in the MPC to compute the input signals to the Arduino robot. Due to reasons further explained in this chapter the physical system was not robust enough to handle the scenarios presented in the simulations, thus two simpler scenarios were tested. These tests were a lane change maneuver and driving past a static obstacle.

7.1 System flow chart

To successfully implement the MPC into hardware multiple parts must work together to achieve the desired result. How the parts are interacting with each other is displayed in Figure 41.

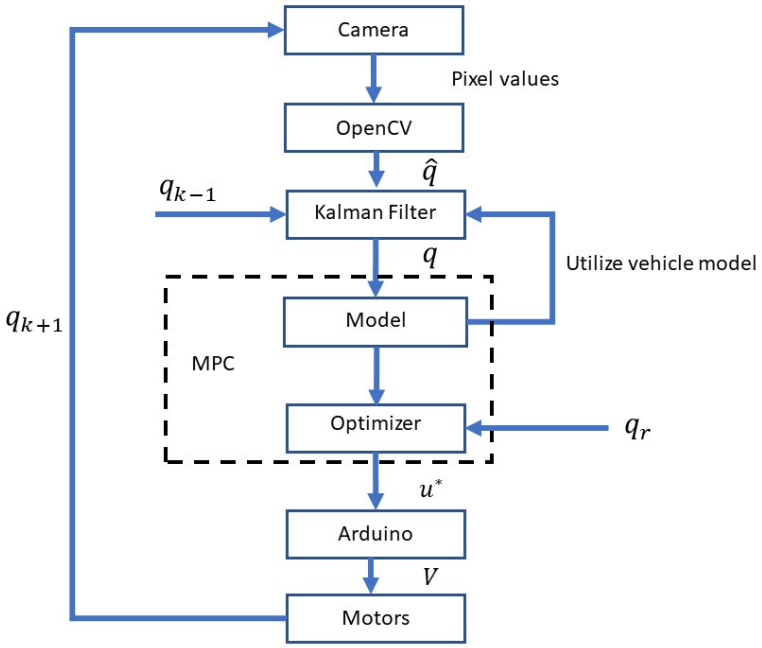

Figure 41: Flowchart of implementation

Firstly, the camera is used to generate the pictures utilized by the camera script to identify the position of the vehicle. Due to inaccuracies and disturbances an Extended Kalman Filter is implemented to generate a more trustworthy estimate of the state. The measured and filtered position signal is then sent to the MPC which computes the optimal trajectory and corresponding control input. The control input is sent to the Arduino via bluetooth which is applied to the DC motors.

7.2 Camera and software

The camera is a Plexgear 720p webcamera. The camera's job is to supply pixel values to the camera script. In the library OpenCV [20] there is a function called HoughCircles which uses the Hough Gradient

Method to identify circular objects. This method looks for points in an image that intersects in the circle's parameter space. If for example the function finds lots of points with the same parameters it means that probably there exists a circle with those parameters. If this probability is higher than some threshold the circle is accepted and shown in the image. This is used to find one big circle that is marked as the rear of the robot and one small that is the front. Using these coordinates the angle and position of the car can be computed. This method is sensitive to noise, even when the background is completely white which eliminates a lot of disturbances. As example the shadow of the car can introduce disturbances that are interpreted as circles. To filter out disturbance outliers were removed by simply checking the distance from them to the center of the previous calculated car center point. If the distance is to far they are neglected. The new accepted circles must also have roughly the same size as the rear or front circle to be accounted for. The output from the camera was also filtered with a low pass filter before the Kalman filter. This was done to filter out some of the disturbances in the camera script which can not be done using the Kalman filter since this is not its purpose.

7.3 Vehicle

The vehicle is a small scale differential drive robot powered by two 6-12 Volt DC motors. The vehicle is controlled by an Arduino micro controller due to it being an easy to use solution. The main task for the Arduino board is to receive the control signals via a bluetooth receiver and apply it to the motors. The bluetooth module is of model HC-05. It was chosen due to it being able to receive as well as sending signals, which allows for eventual future extensions such as applying more sensors to the vehicle. The robot can be seen in Figure 42 and Figure 43. Figure 43 shows the car with the circles attatched which the camera script looks for to estimate the state.

The model described in (29) is not suitable for implementation since the true control inputs are the motor speeds of the right and left motors. To solve this there is a simple relation that can be used to express the model in terms of right and left wheel velocities. The model to be implemented in the real robot will take the form

$$\begin{aligned} x_{k+1} &= x_k + \frac{R_{wheel}}{2}(v_{lk} + v_{rk})\, cos(\theta_k) T_s \\ y_{k+1} &= y_k + \frac{R_{wheel}}{2}(v_{lk} + v_{rk})\, sin(\theta_k) T_s \\ \theta_{k+1} &= \theta_k + + \frac{R_{wheel}}{L}(v_{lk} - v_{rk})\, T_s \end{aligned} \tag{40}$$

where R_{wheel} is the radius of the wheel and L the distance between the two wheels according to Figure 1.

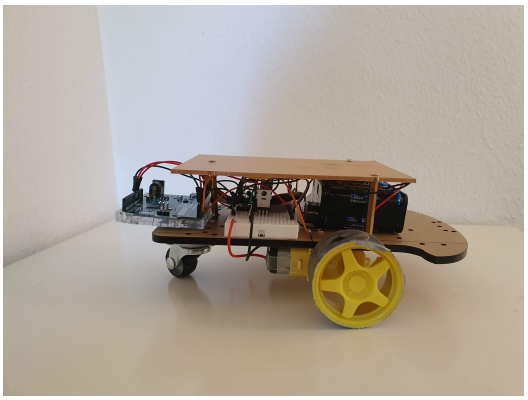

Figure 42: Arduino robot

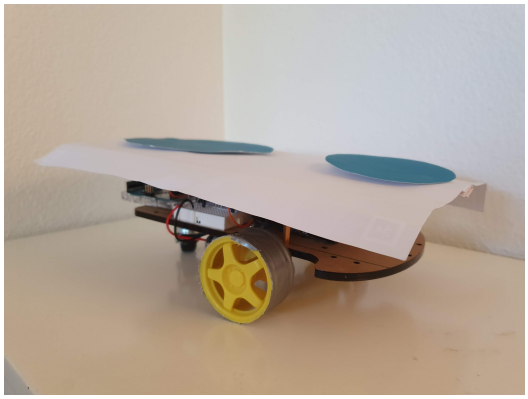

Figure 43: Arduino robot with camera detection circles

The motors of the robot did not respond with precision to the control signals. One engine did most of the time not respond as well as the other to the same input signal. The difference in response to control inputs was not accurately mapped since the system was too stochastic. A scaling factor was added to one of the motors to compensate. Since the motors were inaccurate and the behaviour was stochastic no detailed mapping of velocity and input signal pulls width modulation (PWM) could be made. They were found to roughly follow a decaying cubic function. This was expected as the losses in energy would not scale linearly with the speed, but still this proved too inaccurate.

7.4 Extended Kalman Filter

The Arduino robot as well as the camera script is subject to disturbances and can thus not be trusted completely. This is a perfect case for the Kalman filter. By fusing the model and the measurements the Kalman filter can output results that are more accurate. The chosen Kalman filter is of the type extended Kalman filter, since it's necessary to use a linear approximation of the vehicle model. The Extended Kalman Filter is formulated similarly to the linear Kalman filter, however it utilizes the jacobians to linearize the vehicle model.

$$F_k = \left.\frac{\partial f}{\partial q}\right|_{\hat{q}_{k-1},u_k} \tag{41}$$

$$H_k = \left.\frac{\partial h}{\partial q}\right|_{\hat{q}_{k-1}} \tag{42}$$

Here f is the nonlinear vehicle model presented in (29) which generates the predicted state. The measurement function h describes how the state is measured, which in this case would be the vector

$$h = \begin{bmatrix} x_{camera} \\ y_{camera} \\ \theta_{camera} \end{bmatrix} \tag{43}$$

which when linearized results in a 3×3 identity matrix. For the Extended Kalman filter the prediction is made made as follows. Firstly, the predicted state esimate is:

$$\hat{x}_{k|k-1} = f(\hat{q}_{k-1|k-1}, u_k) \tag{44}$$

And then the predicted covariance estimate is:

$$P_{k|k-1} = F_k P_{k-1|k-1} F_k^T + Q_k \tag{45}$$

Here Q_k is a tuneable weight matrix. With the above prediction the filter is updated. Firstly, the residual, the difference between the measured value and the estimated value, is computed as:

$$\tilde{r}_k = z_k - h(\hat{q}_{k|k-1}) \tag{46}$$

The residual covariance is defined as:

$$S_k = H_k P_{k|k-1} H_k^T + R_k \tag{47}$$

Here R_k is a tuneable weight matrix. If the elements of Q_k are greater then R_k the measurements are trusted more than the model and vice versa. The near-optimal Kalman gain can now be computed as:

$$K_k = P_{k|k-1} H_k^T S_k^{-1} \tag{48}$$

Now the filtered state estimate can be computed using information from both the model and the measurements.

$$\hat{q}_{k|k} = \hat{q}_{k|k-1} + K_k \tilde{r}_k \tag{49}$$

Now the covariance can be updated for the next iteration.

$$P_{k|k} = (I - K_k H_k) P_{k|k-1} \tag{50}$$

Thanks to the Extended Kalman filter the disturbances can be some what accounted for, and the MPC recieves more accurate measurements.

7.5 Summary implementation

The optimizer will control a small differential drive Arduino robot. It gets its control signals from the optimizer which runs on a laptop. The cars states is estimated using a camera in the ceiling and these measurements are fused with a extended Kalman filter together with the model.

8 Results from implementation

The Arduino robot managed to drive two simple scenarios. One where it switched lane and another where it avoided an obstacle. In both cases the input signal had to be amplified. This was because the optimizer outputted too low input signals, the reason being that since the model used in the optimizer does not represent the real system well since the motors are too inaccurate. The disturbance of the camera filter in estimate of angle had a variance of about 1-2 degrees or more. This disturbance is quite high and as discussed in Section 5.2 the optimizers are very sensitive for angle disturbance. Compared to the variance used in the simulation in Section 6 was about 0.1 degrees. The two circles are the rear and front circle that the camera uses to find the state of the car.

8.1 Lane change

Four pictures were captured from different time instances which can be sin in Figure 44. This test was made without the Extended Kalman filter. This was because at the time of testing the Arduino robot did not work properly and the response to the input signals were not correct enough so that the camera measurements were a better estimate than the output from the Kalman filter. The input signal were amplified 8 times. After much tuning the car managed to make a few lane changes. The bottle neck here was the car's motors. In these scenarios the optimizer planned quite well even though the state estimate was subject to a lot of disturbances. In Figure 45 one can see that the car drives incorrectly in the end. The reason why the optimizer plans an odd path in the end is because it is not allowed to decelerate to fast and the robot did not respond to the deceleration phase in the end well enough.

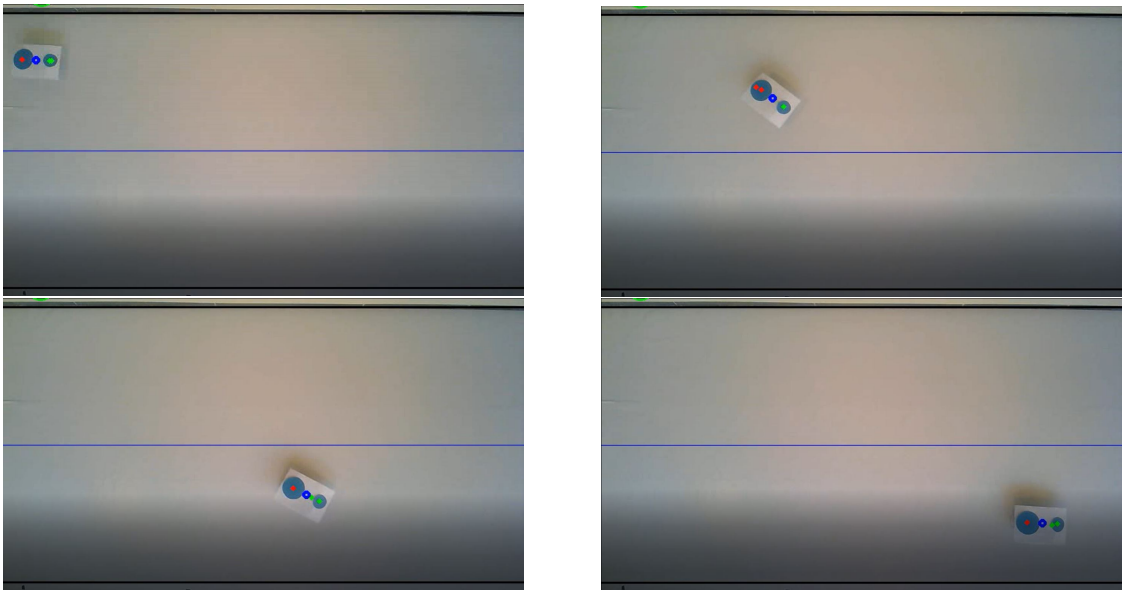

Figure 44: Picture sequence from lane change test

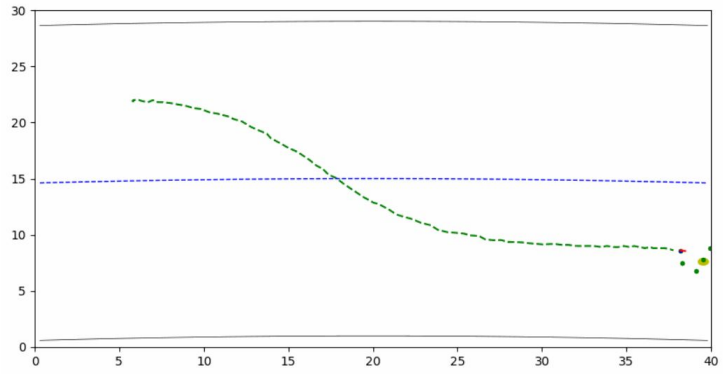

Figure 45: Plot of driven path for lane change

8.2 Obstacle avoidance

Four pictures were captured from different time instances which can be seen in Figure 46. Some constraints were removed to reduce the complexity of the problem on this test. The jerk cost, deceleration constraint as well as the upper border was removed. This was because the optimizer could not converge in this complex environment with the added disturbances from the hardware and the camera measurements.

This was shot using the Kalman filter, but the covariance was still very high on the system dynamics. The covariance ratio was 200/1 for dynamics and camera. The reason that the Kalman was used here was that the Arduino robot responded quite well on the input signals. A small problem with the batteries was solved which made it drive much better. With this modification the test on the lane change could have been improved, but due to lack of time this was not corrected. However, the car model did still not represent the real system well enough to make a remarkable difference.

The main bottle neck in this test was the optimizer. Unlike the lane change test this is a much more constrained test. It was concluded that the optimizer planned bad paths in part due to visual validation from Figure 47, but it could also be observed when running the simulation without the hardware where it worked fine. This means that with the added disturbance the optimizer could not converge. This caused the car to drive on faulty input signals from time to time. Mainly this caused the car to drive too far away from or through the obstacle. This was also the case for this test which is shown in the figures. The car seams to drive good in the pictures just avoiding the obstacle, but in Figure 47 one can see that in fact it did not. This is because the obstacle radius plotted has the ego car radius added to it. The ego car is treated as a point mass. This is how the obstacle avoidance constraint works, see Section 3.4. The reason why it drives odd in the end is partly because the car did not respond to the optimizer correctly, but also that in the very end the camera lost track of the car.

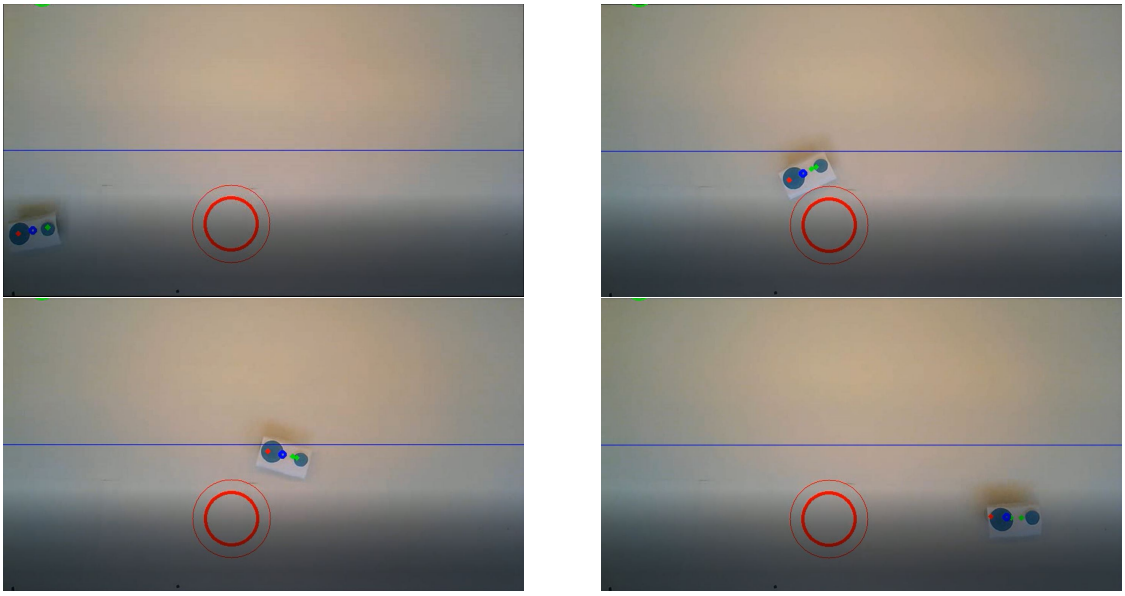

Figure 46: Picture sequence from obstacle avoidance test

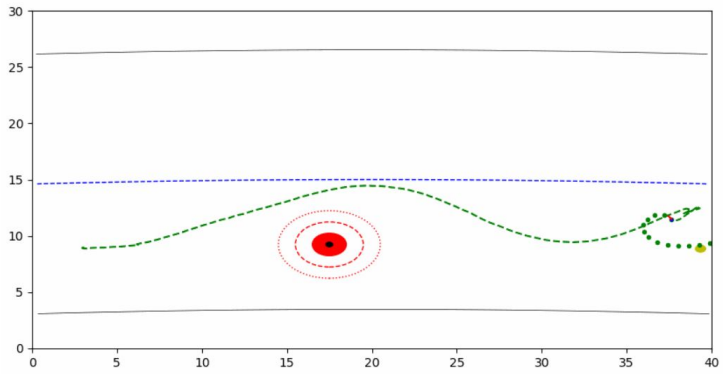

Figure 47: Plot of driven path for obstacle avoidance

8.3 Summary results from implementation

The system that was tested had a lot of disturbances. The camera script introduced disturbances on the states and the model did not mirror the system well since it also had a lot of disturbances. PANOC did not perform well in the tests since it could not converge well when subject to such much noise. The scenarios it was tested in was substantially simplified.

9 Discussion

Three optimizers were evaluated on three different scenarios and they performed differently. The result from both the simulation results and the implementation that was made is discussed below. Possible reasons for specific behaviours and strengths are discussed. Possible areas where each optimizer should be used are also presented.

9.1 Warm start

The importance of warm start in SQP and IP shows how bad they are at global convergence from an arbitrary starting point. Claims of global solutions has been made on both SQP and IP [1] but the assumptions made are restrictive and can be hard or impossible to follow if one would build a very complicated problem. Demonstrated by this rather simple example under Section 6.2 the optimizers cannot converge globally. This project has however not made any big effort in tailoring the algorithms to the specifications needed for global convergence. In many cases in optimization convergence can only be guaranteed in a neighborhood of the solution. This is a well known fact about the newtons method which is used in both SQP and IP for line search. This condition is much easier fulfilled if the optimizer is warm started as the initial guess generally is a guess in the neighborhood of the solution. This is why it can be essential to warm start.

One very interesting comparison between SQP, IP and PANOC is that the latter does not perform as good if warm started. As shown in Figure 9 PANOC converges to a local minimum when warm started which happens very often in these scenarios used by this project. Why it chooses to do so is because it converges fully to this local solution within the given tolerances. It finds a minimizer that are within the tolerances. To tweak the parameters so that this would not happen proved difficult and in many cases without success. The question that arises is how easy is it to meet the criterias for global convergence and can it be done in a complicated environment?

Another thing that is favorable with PANOC is that it converges fast both with and without warm start. This demonstrates how well it converges to at least a local stationary point. PANOC might not always find a global minima, but compared to IP and SQP which sometimes doesn't converge at all, it looks like a very promising algorithm for non-linear optimization problems.

9.2 Sequential quadratic programming and Interior point algorithm comparison

The discussion regarding which of SQP or IP is a better optimizer is a hard and diverse topic. There seems to be few cases where it's clear if one would perform better than the other. Many comparisons has been made that quickly becomes outdated since improvements are made on these algorithms every day. One of these examples is that earlier SQP was preferred in cases where warm start was possible since this was not a possible option with IP method, but this has since become a possibility. Now choosing the right algorithm not that simple. The best way to find out which one is better seems to be testing both of them on the given problem.

The IP algorithm has been claimed to work better for big scale problems as it works on the interior of the problem unlike SQP which works on the null space of the active constraints. One trend is that the

SQP method work better in highly constrained cases since it works on the borders of these which can give it an easier path to the optimum. It outperforms IP in many cases if the ratio $\frac{num.\ \ constraints}{num.\ \ variables}$ is high[1]. One such claim is made by [19] who did tests which proved this. Therefore a similar comparison was made for this project. The problems in Section 6 has 100 decision variables and 348 constraints. When running the optimizers on the problems IP has about twice as many active constraints. This is measured with the median of all the non-zero Lagrange multipliers. Therefore according to [19] IP should perform significantly worse. When looking at the computation time graphs under Section 6 it looks like that this is not the case. They seem to perform similarly in terms of computation time. However, this is not clear in the plots, but IP does perform a bit worse than SQP. It drives the car much slower from start to finish which means that even though each computation is almost the same as for SQP in terms of computation time it does not converge as much as SQP does. Furthermore, the paths seems to be of less quality. This might be because of that the optimal solution lies on a inequality constraint. For example the optimizer tries to put v to v_{max}. IP keeps the solution away from the inequality constraints due to the barrier functions. To converge closer to the constraints the algorithm needs to compute much longer to get a sufficient decrease in the penalty parameter. The results could therefore be less accurate than the SQP method for the same computation time if the solution lies near a highly constrained area[21]. This also makes the IP method sensitive to solutions near constraints as it becomes ill conditioned. This is something that was also experienced in this project were IP seemed to be more sensitive to correct hyper parameters and penalties set on the environment. The problems in this project are quite constrained and especially in some areas as in the scenario *Overtaking maneuver of two moving obstacles* when the car needs to drive in between two cars where the gap is closing. This was see in Section 6.

9.3 Proximal Averaged Newton-type method for Optimal Control

A deeper comparison of PANOC and other optimizers was not found. The comparison has to be made with experiments in this project and comparing the operators in the algorithms which are well studied.

In optimization it is very common to use some variant of Newton's method for the line search which SQP and IP uses in this project. These methods are very good at solving smooth mildly constrained functions. The forward backward splitting method is a general method, which is good at solving non-smooth, constrained and large-scale problems. Evaluating the proximal operator and minimizing the small convex optimization problem, see Section 2.4, often admits closed form solutions [22]. However they often converge slowly to stationary points which may be the reason why it has not been used in the same extent as SQP and IP. PANOC has, as described in Section 2.4, found a clever way to combine this with quasi-newton steps to achieve a much faster and more robust convergence.

PANOC was not very reliable when it came to global convergence and in some cases even local convergence. It outputs more bad paths than both SQP and IP. PANOC is very dependent on a well tuned penalty and is very sensitive to high penalties. In some areas or given some disturbances the car can end up in highly constrained areas where the penalty explodes which might explain why it sometimes does not converge at all or not even locally. Why it is bad at converging globally is harder to explain. Attempts were made to get a more reliable and predictive result through giving the algorithm more time and iterations. No sweet spot was found and letting it compute for longer time, odly enough, often resulted in worse paths.

PANOC seems to have a problem converging optimally in these environments. Even in simpler en-

vironments it showed the same behaviour. One possible theory why this happens is that the step $\hat{u}$ in (19) is a bad estimate for the smooth and non-smooth parts combined, $h + g$, see Section 2.4. Once the projection onto the feasible space is made the step is reduced significantly. This theory is backed by the fact that the number of iterations in PANOC are a lot more than for the other algorithms, see Section 6.2. It may need this many iterations since the projection onto the non-smooth part g keeps reducing the step. Proximal methods sit at a higher level of abstraction than classical optimization algorithms like Newton's method. This is since the shape of the non-smooth part g is not known and are therefore hard to minimize. SQP for example does not use the non-smooth parts explicitly but aproximates them and this has shown to work quite well if the optimizer is close to a solution. Projecting the step given by h onto g does not mean it is a suitable step for g. It might be so that projecting onto g keeps it from reaching an optimal point by bouncing it around the minimum. This could explain why PANOC does not get a better result when it is allowed to do more itterations. It is known that the proximal operator is sensitive to ill conditioning [22]. If g is ill conditioned around $\hat{u}$ then projecting it onto g would clearly be a hard thing to get right.

Why PANOC sometimes get worse with more iterations could be explained with that since the objective function keeps changing due to that the penalties and the λ variables changes for each outer iteration. This is only explains it partly. At some point given enough iterations it should converge given a significant small step size, but this is not the case. In PANOC it's not possible to change the step size and thus this theory could not be tested. However, no deeper investigation was made on the criteria that the authors of PANOC stated for global convergence. SQP and IP on the contrary gives better results when allowed to compute longer, which is the expected behaviour.

9.4 Recommended use

PANOC is best used in time critical systems due to its superior convergence rate. It can also be used in highly constrained environments and even non-smooth and nonlinear ones. It should not be used in these difficult environments if the system is unstable or if the requirements on optimal output is high since it is not reliable. The output must be checked so that it meets the specified requirements PANOC is a good choice when there is no way to warm start the optimizer with a good initial guess.

SQP is best used in medium to highly constrained environments that are not time critical. Much computation time can be given which improves the quality of the solution and is therefore good for systems that are unstable or the need for optimal solution is high. For mildly constrained environments the need for warm start is not that high. The algorithm can be used in nonlinear, non-smooth environments given that it is warm started with a guess close to the solution.

IP is not recommended for medium to highly constrained environments. It should not be used in very non-linear or non-smooth environments. If the solution is located close to or on difficult constraint borders IP will have a hard time converging, which could imply that another optimizer would be preferable in those cases.

9.5 Collision avoidance system

The cone approach for obstacle avoidance was effective. It allows the ego vehicle to effectively plan ahead a path so that the next control input takes it to a place where it is less likely to collide. The non-linear

cone seems to not slow down computation speed in either SQP, IP or PANOC compared to the linear collision cone. This was concluded as they performed as well when the scenario *Overtaking maneuver of two moving obstacles* was simulated with obstacles on the intersection points with 0 velocity which is essentially how the non-linear collision cone works. It does not treat dynamic obstacles as dynamic, but static obstacles projected forward in time. This is why this comparison and conclusion could be made.

9.6 Model predictive control in autonomous vehicles

SQP and IP are not good candidates for implementation in time critical systems or systems with low computational power. They will not be fast enough and the environment modeling has to be restricted substantially to ensure that they converge. PANOC does show great promise since it seems to handle difficult environments quite well and is fast enough. It is still quite sensitive to the magnitude of disturbances that could be expected in real life. The environment should also be made as simple as possible as PANOC is not reliable in complex environments. In very controlled settings PANOC could work well if safety is not of concern.

There is a lot of papers that uses MPC to drive cars in simulations. One might get the impression that it is a perfect solution, but it still has many flaws. For example it is hard to imagine MPC in a robot like lawn mowers. The reason is that the environment is simply to complex to model. For PANOC to work in such a setting, the environment has to be modelled in a clever way that reduces its complexity.

PANOC is groundbreaking in the field of non-linear optimizers. While it may not yet be able to tackle hard problems with robustness the opportunities it opens are wast. The improved computation time and the promise it has shown on convergence in non-smooth and highly non-linear environments is impressive.

9.7 Ethics and the environment

Autonomous vehicles is a topic where ethical questions are ever present. How can a computer take actions that don't cause humans to come to harm. One way this is taken into account in this project was by not having border constraints as hard constraints as mentioned in Section 3.5. The reason for not having hard constraints on the border was an active choice. Firstly the argument could be made that the vehicle should not be allowed to leave the road, since this would mean a loss in resources due to damaged equipment. This is a valid point, but it raises another question, if the situation occurs would it be better to save the equipment or people's lives? Assume that the ego vehicle is autonomous without any human supervision, further assume that it is driving in factory environment. If a person would suddenly appear in the path of the vehicle it's fair to argue that the vehicle should try to save the person rather than save itself. So this is highly a question of ethics. Simply put, not all scenarios where it's possible to leave the road are bad ones. There could definitely occur scenarios when it's preferable for autonomous vehicle to leave the road rather than crash into obstacles or persons. Consider the scenario when the vehicle is going at max speed and a suddenly appearing obstacle appears. Making a panic maneuver could lead to devastating consequences and loss of life. In this project the following has been considered; obstacles could be considered humans and collision with these should thus be avoided with all means possible. The vehicle leaving the road and crashing into a ditch or a wall is, with this mentality, considered preferable. This is the primary reason for using soft constraints on the road boarder in this project.

Further, with MPC one could choose to optimize towards minimization of energy consumption. The vehicle could therefore be able to drive further on less energy. This has the potential to greatly improve the emissions vehicles cause and therefore help the environment.

9.8 Summary of discussion

It was found that warm start is essential for IP and SQP to converge, while PANOC is less effected by this. PANOC see some benefits from cold-start since it avoids local minimas if cold-started.

PANOC was the superior in regards to speed, followed by SQP and lastly IP. Further, IP doesn't converge to solutions located on a inequality constraint, while SQP and PANOC have no problem converging to a reference state.

PANOC is best for time critical systems, since it's very fast. Albeit unreliable in some instants. SQP is best used in highly constrained environments that are not time critical, since it's slower than PANOC. IP is not recommended in this type of scenarios since it has a hard time converging to reference states placed on the constraints, further it's also the slowest algorithm. IP could work well if the system has a lot of decision variables.

Complex environments makes it hard to recommend MPC for use in self-driving vehicles. For vehicles driving in simple environments MPC could however be a valid choice, especially with algorithms like PANOC which is fast enough for embedded applications.

10 Conclusion

This project evaluated the optimizers sequential quadratic programming (SQP), interior point method (IP) and Proximal Averaged Newton-type method for Optimal Control (PANOC) in realistic scenarios for control of a vehicle on a road. The scenarios were simulated and then PANOC was further evaluated on an Arduino robot.

PANOC had a superior computation speed and converge more often than the others in highly constrained, nonlinear, non-smooth environments. This is most probably because of the proximal operator used in the algorithm that converges better than the quasi newton method used in SQP and IP in these nonlinear and non-smooth environments. It is almost performing equally well with and without warm start and handles disturbances good, demonstrating its superior convergence ability on these types of problems. However it does not converge optimally often and is prone to get stuck in local minimas.

SQP and IP converges more optimally and with much higher robustness if warm started close to the optimal solution. They are very slow compared to PANOC and very sensitive to disturbances when the environment is highly constrained, nonlinear and non-smooth. They must be warm started with a guess close to the solution to work. They are also very dependent on the correct tuning of constraints. SQP does however, converge to a more optimal solution than the others, when it's given enough time. In more mild environments SQP and IP can be cold started, require less time to converge, be more robust and converge more optimally.

PANOC should be used in systems which are time critical and if it is highly constrained it must be a stable system since PANOC is not predictable. SQP and IP should be used in milder environments which are not time critical. If the environment is highly constrained they must be warm started with a good initial guess. SQP is preferred over IP in environments with medium to high number of constraints compared to the number of decision variables[19].

From this thesis a lot of doubt has been raised whether or not optimizers are good enough for implementation in autonomous vehicles if the environment is complex, stochastic or nonlinear/non-smooth. If MPC shall be used in self driving vehicles the setting it drives in should be controlled or simplified and the system should be stable. Furthermore, precautions must be taken to validate that the output from the optimizers have converged in a desirable manner.

11 Future work

One of the biggest issues was the bad and unreliable convergence. It would be interesting to put more emphasise on global convergence guarantees for PANOC and investigate what limitations this will pose on the problem if implemented. PANOC had a very odd behavior when it was allowed to compute for longer periods of time. When the computation time were increased it performed a lot worse. No reason for this behavior was found. If this is solved one could sacrifice some computation time for accuracy just like one can do with IP and SQP which in many cases would be preferable.

It might be hard to improve on the optimizers convergence capability. The chance of global convergence is increased with reduced complexity of the modeled environment. It would therefore be interesting to see if it could be modeled in another way. Could some parts of the environment be handled by other sensors to ease the job of the optimizer? Could some sort of method fusion or sensor fusion be used so that the optimizer has a milder environment to work with?

It would also be interesting to further evaluate PANOC's performance in real time applications with better hardware. This would give an even better understanding of the very promising algorithm by eliminating some hardware related error sources. Those error sources were inaccuracies in the hardware that could not be modeled.

Lastly more variety of simulation problems could be developed. If a deeper knowledge and understanding of these problems a more accurate evaluation and comparison could be made.

References

[1] Jorge Nocedal and Stephen J. Wright. *Numerical Optimization*. Springer, New York, NY, USA, second edition, 2006.

[2] Zulfiqar Habib Iram Noreen, Amna Khan. A comparison of rrt, rrt* and rrt*-smart path planning algorithms. *IJCSNS*, 16, 2016.

[3] Dustin J. Webb and Jur van den Berg. Kinodynamic rrt*: Optimal motion planning for systems with linear differential constraints. *CoRR*, abs/1205.5088, 2012.

[4] Marcos R. O. A. Maximo. Model predictive controller for trajectory tracking by differential driver-obot with actuation constraints. *Simposio Brasileiro de Automa ao Inteligente (SBAI)*, 2015.

[5] L. Stella, A. Themelis, P. Sopasakis, and P. Patrinos. A simple and efficient algorithm for nonlinear model predictive control. In *IEEE Conference on Decision and Control (CDC)*, pages 1939–1944, Dec 2017.

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

[7] embotech. https://www.embotech.com/. Accessed: 2020-03-27.

[8] C. Liu, S. Lee, S. Varnhagen, and H. E. Tseng. Path planning for autonomous vehicles using model predictive control. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 174–179, 2017.

[9] A. K. Singh and K. M. Krishna. Reactive collision avoidance for multiple robots by non linear time scaling. In *52nd IEEE Conference on Decision and Control*, pages 952–958, 2013.

[10] Mithun Babu, Yash Oza, Arun Kumar Singh, K. Madhava Krishna, and Shanti Medasani. Model predictive control for autonomous driving based on time scaled collision cone. *CoRR*, abs/1712.04965, 2017.

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

[12] H. H. Rosenbrock. An Automatic Method for Finding the Greatest or Least Value of a Function. *The Computer Journal*, 3(3):175–184, 01 1960.

[13] J.B. Rawlings and D.Q. Mayne. *Model Predictive Control: Theory and Design*. "Nob Hill Publishing", 2009.

[14] Shih Han. A globally convergent method for nonlinear programming. Technical report, USA, 1975.

[15] M. J. D. Powell. Variable metric methods for constrained optimization. In R. Glowinski, J. L. Lions, and Iria Laboria, editors, *Computing Methods in Applied Sciences and Engineering, 1977, I*, pages 62–72, Berlin, Heidelberg, 1979. Springer Berlin Heidelberg.

[16] Constrained nonlinear optimization algorithms. https://se.mathworks.com/help/optim/ug/constrained-nonlinear-optimization-algorithms.htmlbrdsjhj. Accessed: 2020-03-30.

[17] Joel A E Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. CasADi – A software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

[18] Obstacle avoidance in a dynamic environment: A collision cone approach. https://ieeexplore.ieee.org/document/709600. Accessed: 2020-02-19.

[19] B Marteau J Fiala. Nonlinear optimization: A comparison of two competing approaches. https://www.nag.co.uk/market/nonlinear-optimization-comparison.pdf. Accessed: 2020-03-19.

[20] G. Bradski. The OpenCV Library. *Dr. Dobb's Journal of Software Tools*, 2000.

[21] Choosing the algorithm. https://se.mathworks.com/help/optim/ug/choosing-the-algorithm.html. Accessed: 2020-05-19.

[22] Proximal algorithms. https://web.stanford.edu/ boyd/papers/pdf/prox$_a$*lgs.pdf*. *Accessed* : 2020 − 05 − 19.

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

CHALMERS
UNIVERSITY OF TECHNOLOGY