Erik Thallaug Fagerli

Model Predictive Control for Path Following of an Autonomous Student Car

Model Predictive Control for Longitudinal and Lateral Path Following of an Energy Efficient Autonomous Student Car

Master's thesis in Industrial Cybernetics
Supervisor: Thor Inge Fossen
June 2022

Erik Thallaug Fagerli

Model Predictive Control for Path Following of an Autonomous Student Car

Model Predictive Control for Longitudinal and Lateral Path Following of an Energy Efficient Autonomous Student Car

Master’s thesis in Industrial Cybernetics
Supervisor: Thor Inge Fossen
June 2022

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Abstract

DNV FuelFighter has been a part of the competition *Shell Eco-Marathon* since 2008. The competition revolves around creating the most energy-efficient car. In 2018 Shell Eco-Marathon launched their first autonomous competition where each team needed to create an autonomous car. DNV FuelFighter decided in 2020 that they wanted to compete in the autonomous competition. For an autonomous car to work, several subsystems need to work together. One of the subsystems is the control system. Here the vehicle needs to decide what angle and velocity it should have to reach its destination.

This thesis focuses on the car's control system, where a longitudinal and lateral controller is developed and implemented. DNV Fuelfighters autonomous car must manage to follow a given path while being energy efficient and following a reference velocity. A Model Predictive Controller (MPC) has been developed to achieve this. The MPC will give the optimal steering angle and velocity such that the different criteria are fulfilled. Two MPCs have been developed to compare energy efficiency, one that includes the energy criteria (Energy MPC) and one that does not (Non-Energy MPC).

The model used to describe the vehicle is a combination of both kinematic and dynamic bicycle models, where the dynamic model is used for longitudinal control and the kinematic model is used for lateral control.

Two scenarios were tested to compare the two MPCs, where performance, energy efficiency, and computational load were the criteria. The results showed that both MPCs managed to follow the given path, but the Energy MPC saved around 12% energy. However, this came with the cost of reduced velocity. The Energy MPC was then implemented with ROS and Gazebo and simulated under a more realistic environment. The result from this simulation showed good performance from the Energy MPC.

Sammendrag

Siden 2008 har DNV FuelFighter vært en del av konkurransen Shell Eco-Marathon. Konkurransen dreier seg om å lage den mest energieffektive bilen. I 2018 lanserte Shell Eco-Marathon sin første autonome konkurranse. 2020 var det første året DNV FuelFighter bestemte seg for å konkurrere i den autonome konkurransen. For å lage en autonom bil, må flere delsystemer fungere sammen. Et av delsystemene er kontrollsystemet. Her må kjøretøyet bestemme hvilken vinkel og hastighet den skal ha for å nå målet.

Denne oppgaven fokuserer på kontrollsystemet til bilen, hvor en langsgående og lateral kontrollsystem er utviklet slik at den får DNV Fuelfighters autonome bil til å følge en gitt vei samtidig som at den er energieffektiv og følger en referansehastighet. For å oppnå dette er det utviklet en Model Predictive Controller (MPC). MPCen vil gi optimal styrevinkel og hastighet slik at de ulike kriteriene er oppfylt. For å sammenligne energieffektivitet er det utviklet to MPCer, en som inkluderer energikriteriet (Energy MPC) og en som ikke gjør det (Non-Energy MPC).

Modellen som brukes for å beskrive kjøretøyet er en kombinasjon av både kinematisk og dynamisk sykkelmodell hvor den dynamiske modellen brukes for langsgående kontroll og den kinematiske modellen brukes for lateral kontroll.

For å sammenligne de to MPCene ble to scenarier testet hvor ytelse, energieffektivitet og beregningstid ble evaluert. Resultatene viste at begge MPC-ene klarte å følge den gitte banen, men Energy MPC-en sparte rundt 12% energi. Dette kom med kostnadene at den kjørte med redusert hastighet. Energy MPCen ble deretter implementert sammen med ROS og Gazebo og simulert under et mer realistisk miljø. Resultatet fra denne simuleringen viste at MPCen hadde fortsatt god ytelse.

Preface

This thesis represents my work throughout the year 2021/2022 as part of DNV FuelFighter The thesis is based of the previous years specialization project [1] where an Adaptive Model Predictive Control (AMPC) was tested with a dynamic bicycle model. The model used in this thesis is a combination of both kinematic and dynamic bicycle model, and the AMPC has been converted to a non-linear MPC. I want to give a huge thanks to DNV FuelFighter, especially the autonomous group, for allowing me to be a part of the team and contribute to DNV FuelFighter's first autonomous car. Unfortunately, due to unforeseen circumstances, I will not be able to participate in the autonomous competition held in France, but I am confident that the team will represent NTNU well. I hope my work will be a stepping stone for future members and be used to develop the system further.

I would also like to thank Thor Inge Fossen for being my supervisor. Thor I. Fossen had helped shape my thesis and always been available when I needed guidance. Lastly, I would like to thank my friends and family for trusting in me and giving me the motivation I needed to push through.

Contents

Figures

Tables

Code Listings

Chapter 1

Introduction

The usage of automation has become a central part of our environment. With the help of autonomous robots, jobs revolving around mundane, time-consuming, and dangerous tasks have been replaced with robots. Automation has led to increased interest in robotics, and one of the fields that have seen tremendous interest is automation in the vehicle industry [2]. Autonomous Ground Vehicles (AGV) are one of the fields rising in popularity and will also be the main focus of this paper.

As autonomous cars are becoming a more significant part of society's infrastructure, competitions around the globe are being held to see who can produce the best autonomous car. One of these competitions is the "Shell Eco-marathon". Shell Eco-marathon is a global academic program focused on energy optimization of cars [3] and, as of 2018, created an autonomous competition. The student organization DNV FuelFighter [4] intends to compete in this competition this year. DNV FuelFighter consists of several subgroups, such as mechanical, electrical, software, and design, and as of 2020, created the subgroup autonomous. The autonomous group focuses on the autonomous part of the vehicle, with the combination of ROS (Robot Operating System, [5]), sensor, perception, path planning, and control.

For a vehicle to drive autonomously, it must know what steering angle and velocity it must output for the vehicle to reach the given goal. One way to achieve this is by using Model Predictive Control (MPC). MPC is an advanced control technique used for multi-variable control problems [6]. The use of MPC to control autonomous cars is something that has been researched quite thoroughly and will be discussed in Section 1.5 and Chapter 2.

1.1 Shell Eco Marathon

As mentioned earlier, Shell Eco-Marathon is a global academic program that focuses on energy optimization. The competitions are arranged all over the world. The autonomous competition will be held from the 3rd to the 6th of July in Nogaro, France. The competition consists of four different challenges, where each team can score up to 25 points for each challenge. The challenges are:

- Driving Autonomously
- Parking Maneuverability
- Obstacle Avoidance
- Business Presentation

Schematics of the different challenges are shown in Figure 1.1 - 1.3. It is important to note that the schematics do not represent the final challenge and track layouts but indicate what to expect at each challenge. The first challenge consists of driving around a track. The goal is to drive around the track without hitting the borders, there will be no extra points for energy efficiency, but the car's performance will be evaluated. The second challenge is parking maneuverability. The track will contain multiple marked squares where only one of the squares is possible to park in. The goal is to find that square and park inside of it. The third challenge is obstacle avoidance. Here the car must drive between different obstacles without touching them. The last challenge is a business presentation. The judges will be looking at how well the autonomous car performed at the other competitions, the quality of the autonomous system, finances, and the consideration of energy efficiency. The team with the most points after the four competitions will receive a prize of 2500 euros.

Before the car is allowed to compete, it must first pass a test where the judges inspect the car and make sure that it follows the guidelines given.

DNV Fuelfighter also participates in a manned car competition. Here the only goal is to be as energy-efficient as possible around a particular track. This year the manned car competition will be held in Assen, Netherlands. This report will not further detail this competition since the report only focuses on the autonomous car.

1.2 DNV FuelFighter

DNV FuelFighter is a student organization that has been part of the Shell Eco Marathon competition since 2008. The team has consistently preformed well in these competitions and plans to also do it this year. The autonomous group was first created in 2020, but due to Covid-19, they did not have the opportunity to compete. This is the first year the autonomous group will be able to test their autonomous car. Since the group is fairly new their main goal is to make the car drive autonomously without a big consideration of energy efficiency. To achieve this several parts need to work together. This is all explained in Appendix A. The goal of the autonomous group is to complete each of the different challenges in a safe and efficient manor. To achieve this the controllers demand accuracy and robustness. For the car to achieve this it is crucial to research both longitudinal and lateral control that allows the car to reach its goal. With the help of MPC, DNV FuelFighter believe that this can be accomplished. The MPC will embedded into the car FuelFighter 5.

1.3 Key Contributions

This thesis is a contribution to DNV Fuelfighter's autonomous vehicle. The following contributions have been made:

- Literature study on vehicle models and MPCs
- Development and implementation of the kinematic bicycle model in Python
- Development and implementation of two MPC controllers, one that includes energy minimization and one that does not.
- Implementation and testing in Python and Gazebo
- Discussion of the performance of the two MPCs

1.4 Problem formulation

This thesis aims to create an MPC for longitudinal and lateral path-following with multiple objectives. The objective of the MPC is to track a given path given, follow a reference velocity and minimize energy consumption. The MPC will the named *Energy MPC*. To validate energy minimization, an MPC will be implemented that focuses only on following a given path and reference velocity but does not focus on energy minimization. This MPC will be called *Non-Energy MPC*.

Some assumptions have to be made such that the scope of the thesis narrows down. The assumptions made through this thesis are

- No prior knowledge of the whole track the car will drive
- Limited CPU usage
- Limited turning radius and speed
- Limited sensors
- Low velocity

1.5 Related Work

The use of MPC to control both lateral and longitudinal movement of vehicles has already been researched. A study done by [7] compared the use of a kinematic and dynamic model in combination with MPC to see the effectiveness of each model when under the influence of windy roads. The paper concludes that kinematic models are better suited for lower speeds, but trajectory tracking errors increase at higher speeds. For higher speeds, the dynamic model works better. A study done by [8] compared the performance of a kinematic model vs. a nine degrees-of-freedom model when it came to trajectory planning. It concluded that the kinematic model was valid under a certain lateral acceleration and at high acceleration; a more valid model should be implemented.

A paper done by [9] showed how energy consumption could be reduced by integrating it into the cost function of an MPC. They compared their MPC with a PID and showed that it could save up to 1.27% energy when following natural measured road conditions. [10] showed even better results in energy savings when they compared two different MPCs, a sports mode MPC that focused on reaching the goal as fast as possible and an energy savings MPC that focused on minimizing energy consumption. They concluded that the energy-saving MPC used, in the best case, 48% less energy than the sport mode MPC. A study done by [11] showed that economic MPC could be used to minimize energy consumption. The paper shows that economic MPC can be used to solve real-time iterations with minimal time. [12] introduced the idea of how multiple vehicles can coordinate together to minimize energy consumption in an intersection.

1.6 Structure of the report

Vehicle Modelling

This chapter gives insight into how the vehicle model is created. This model is used to describe how the autonomous vehicle behaves. The model is used to describe lateral and longitudinal behavior.

Control System Design

Theory about MPC is introduced, and how energy efficiency is incorporated into the MPC. The problem formulation is presented with the necessary constraints and minimization variables.

Implementation

An overview of the software and framework used to solve and simulate the MPC is presented. Further implementation in a more realistic environment and the algorithm used to describe path representation is introduced. Lastly, the parameters used to describe the vehicle are shown.

Results

Here the results of the MPC will be discussed. Three different scenarios will be tested.

- straight-line driving,
- sinus wave path, and
- path in Gazebo.

Straight-line and sinus waves will be tested twice, once with Energy MPC and once with Non-Energy MPC. These will indicate the performance, energy consumption, and computational load of the two MPCs. The Energy MPC will be the only one simulated in the Gazebo.

Discussion

Performance, energy consumption, and computational load of the two MPCs will be discussed.

Conclusion and Future Work

A conclusion of the MPCs performance will be presented, and lastly, a discussion of what can be further implemented to enhance the performance of the MPC.

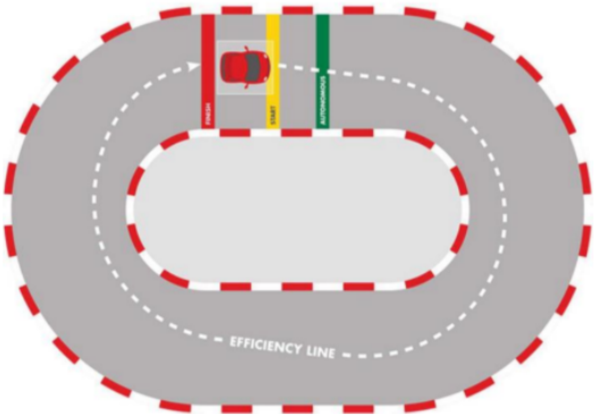


Figure 1.1: Schematic: Autonomous driving (source: [13])

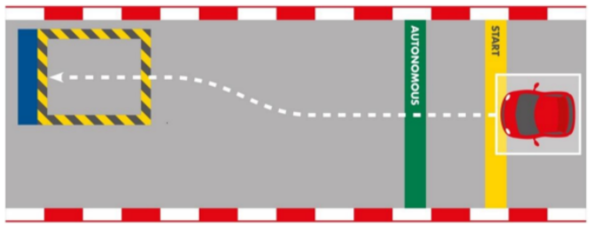


Figure 1.2: Schematic: Parking maneuverability (source: [13])

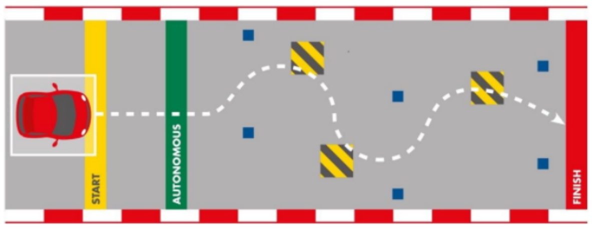


Figure 1.3: Schematic: Obstacle avoidance (source: [13])

Chapter 2

Vehicle Modelling

How a vehicle is modeled can significantly impact how the system behaves. In this chapter, the theory behind how the vehicle model is structured is presented in Section 2.1. In this section, the different equations used to describe the vehicle are presented. A summary of the vehicle model is presented in Section 2.2.

2.1 Vehicle Model

There are many ways of describing how a vehicle behaves. The two most common ways are the kinematic model and dynamic model. The kinematic model describes the motion of the vehicle based purely on the geometric relationships governing the system. In contrast, the dynamic model describes the motion of the vehicle with respect to forces and motion [14]. When describing the motion of a vehicle, it is common to separate longitudinal motion and lateral motion.

2.1.1 Longitudinal Vehicle Dynamics

Looking at the external forces that affect the vehicle in Figure 2.1, the forces can be categorized into; aerodynamic drag forces, gravitational forces, longitudinal tire forces, and rolling resistance forces, as shown in Figure 2.1. Describing these forces is Newton's second law. The force balance law yields

$$m\ddot{x} = F_{xf} + F_{xr} - F_{aero} - R_{xf} - R_{xr} - mg\sin(\theta) \tag{2.1}$$

where

- F_{xf} is the longitudinal forces of the front tires
- F_{xr} is the longitudinal forces of the rear tires
- F_{aero} is the aerodynamic drag forces
- R_{xf} is the rolling resistance force of the front tires
- R_{xr} is the rolling resistance force of the rear tires
- m is the mass of the vehicle
- g is the acceleration due to gravity

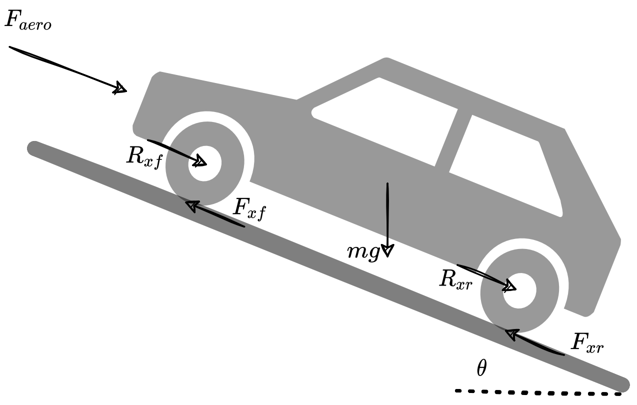


Figure 2.1: Longitudinal forces acting on the car

- θ is the angle of inclination

The aerodynamic drag force can be further expressed as

$$F_{aero} = \frac{1}{2}\rho C_d A_f (v + v_{wind})^2 \tag{2.2}$$

where

- ρ is the density of the air
- C_d is the aerodynamic drag coefficient
- A_f is the frontal area of the vehicle
- v is the longitudinal velocity of the vehicle
- v_{wind} is the longitudinal velocity of the wind (positive when headwind and negative when tailwind)

Every parameter except C_d can easily be obtained. One way to obtain C_d is by using a coast down test. In this test there no throttle or angle inputs and assuming $\theta = 0$ and $v_{wind} = 0$. The force balance equation can thus be expressed as

$$-m\frac{dV_x}{dt} = \frac{1}{2}\rho A_f C_d v^2 + R_x \tag{2.3}$$

This equation can be solved for C_d. A detailed explanation of how to calculate C_d can be found in [14].

The longitudinal tire forces F_{xf} and F_{xr} are friction forces from the ground acting on the tires. Experimental results have shown that the tire forces depend on slip ratio, normal load on the tire, and the friction coefficient of the tire-road interface. For simplicity's sake, this paper will assume that these forces are minuscule, and the longitudinal tire forces on the driving wheels only depend on the driveline dynamics. The driveline components usually consist of an engine, a torque converter, transmission, final drive, and wheels for a conventional car. However, in our case, the motors are connected to the wheels, called in-wheel motors. A

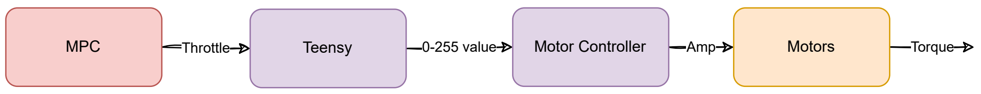


Figure 2.2: Flowchart of the hardware that controls longitudinal dynamics

flowchart of the total driveline components are shown in Figure 2.2. Here it is shown that the throttle input is connected to a Teensy [15]. The Teensy converts the throttle input into a 0-255 value. This value is then sent into a motor controller, which then turns the 0-255 value into amps and is then sent to the motor. The motor is connected to the wheels via gears. The motors used in FuelFighter 5 are two DC motors [16]. With this information, it is possible to describe the tire forces of the vehicle with the throttle as input. Torque can be expressed with either respect to force or respect to effect.

$$T = F \cdot r \tag{2.4}$$

$$T = \frac{P}{\omega} \tag{2.5}$$

where F is tire force, r is radius of the wheel, P is effect, and ω is the angular velocity. P and ω can be further expressed as

$$P = V \cdot I \tag{2.6}$$

$$\omega = \frac{2\pi \cdot N_{out}}{60}, \tag{2.7}$$

where V is voltage input, I is current input, N_{out} is rpm (rotations per minute). It is important to note that N_{out} is the rotation of the wheel and not the motor. To convert the rpm of the wheels to the rpm of the motor, the gear ratio between them must be acquired. The equation for this is

$$N_{out} = \frac{N_{in}}{G} \tag{2.8}$$

where N_{in} is the rpm of the motor and G is the gear ratio. N_{in} can then be converted back to ω by using (2.7). ω can also be expressed as

$$\omega = \frac{v}{r} \tag{2.9}$$

where v is the velocity of the car. Lastly, the input current needs to be described as a function of throttle input. After talking with the electrical subgroup, they informed that the motor-controller current is proportional with the input. Thus meaning that the input current can be described as

$$I = K \cdot throttle \tag{2.10}$$

where K is a constant. By inserting (2.4), (2.6), (2.7), 2.9, (2.8), and (2.10) into (2.5) the equation to describe tire force becomes

$$F = \frac{V \cdot K \cdot throttle \cdot r}{v \cdot G}, \quad v > 0 \tag{2.11}$$

One thing that is important to note with (2.11) is that if the velocity is zero the equation is not defined. To counteract this a constant, ϵ is placed in the denominator. The final expression for tire force is

$$F = \frac{V \cdot K \cdot throttle \cdot r}{v \cdot G \cdot m + \epsilon} \quad \epsilon > 0 \tag{2.12}$$

By inserting (2.12) into (2.1) and considering the assumption that the rolling resistance, wind speed and angle of inclination is negligible, the longitudinal forces is expressed as

$$m\ddot{x} = \frac{V \cdot K \cdot throttle \cdot r}{v \cdot G \cdot m + \epsilon} - \frac{1}{2}\rho C_d A_f v^2. \tag{2.13}$$

The assumptions that rolling resistance and angle of inclination are negligible can be made based on the assumptions made in Section 1.4.

2.1.2 Kinematic Bicycle Model

As mentioned earlier, the kinematic model describes the motion of a vehicle based purely on the geometric relationships governing the system. Figure 2.3. depicts a vehicle model where the two front and rear wheels are combined to create a bicycle model. The angle of the front wheel is represented by δ and it is assumed that the angle of the rear wheel is always zero. The distance between the center of gravity (CG) and the front and rear wheels are l_f and l_r. The velocity is v, and the angle of the velocity is β. This angle is called the slip angle. ψ is the heading angle of the vehicle. Using a bicycle model compared to a 4-wheeled model is that the simplification makes it less computational heavy, but still a fairly accurate model describing the motions. This is shown in [17].

2.1.3 Kinematic Model of Lateral Motion

Under the assumption that the velocity vector in the front and rear wheels point in the direction of the wheels' orientation, the kinematic model shown above is valid. The front wheel makes an angle δ, while the rear wheel is assumed to have an angle of zero. This assumption assumes that the "slip angle" β is equal to zero. The slip angle of a tire is the angle between the orientation of the tire and the orientation of the velocity vector. When the vehicle is traveling straight ahead, the orientation of the wheel and the velocity vector are the same, thus meaning that the slip angle is equal to zero. The assumption that the slip angle is small can be used when the vehicle is traveling at low speeds (less than 5 m/s). This is a reasonable assumption because the lateral force generated by the tires is small

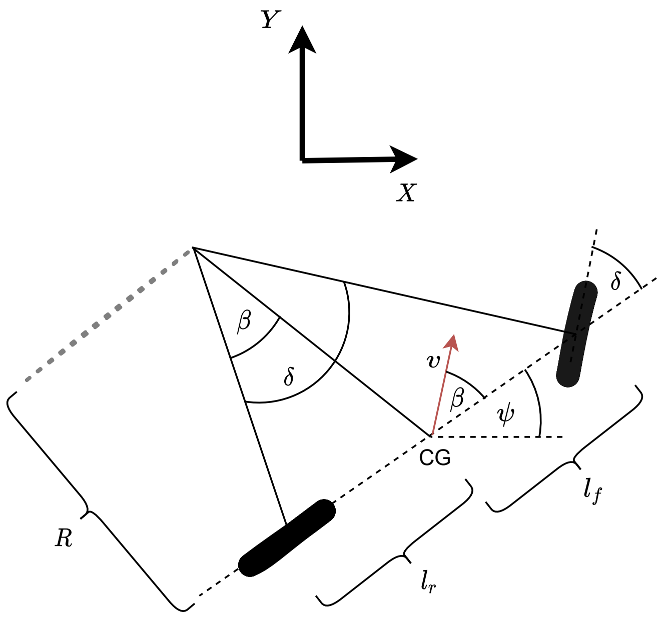


Figure 2.3: Bicycle model

when driving at low speeds. This is further proven from (2.14) which shows that the lateral forces are low when the velocity is low.

$$F = \frac{mv^2}{r} \tag{2.14}$$

By applying the sine rule to the front triangle, we get the expression

$$\frac{\sin(\delta - \beta)}{l_f} = \frac{\sin(\frac{\pi}{2} - \delta)}{R}. \tag{2.15}$$

(2.15) can be extended to

$$\frac{\sin(\delta)\cos(\beta) - \sin(\beta)\cos(\delta)}{l_f} = \frac{\cos(\delta)}{R}. \tag{2.16}$$

By multiplying (2.16) with $\frac{l_f}{\cos(\delta)}$ we get

$$\tan(\delta)\cos(\delta) - \sin(\beta) = \frac{l_f}{R}. \tag{2.17}$$

This procedure is also done with the rear wheels

$$\sin(\beta) - \tan(\delta_r)\cos(\beta) = \frac{l_r}{R}. \tag{2.18}$$

Adding (2.17) and (2.18) results in

$$(\tan(\delta) - \tan(\delta_r))\cos(\beta) = \frac{l_r + l_f}{R}. \tag{2.19}$$

Under the assumptions that the radius of the path of the vehicle changes slowly, the rate of change in orientation ($\dot{\psi}$) is equal to the angular velocity

$$\dot{\psi} = \frac{v}{R} \tag{2.20}$$

By inserting (2.19) into (2.20), (2.20) can be rewritten as

$$\dot{\psi} = \frac{v\cos(\beta)}{l_f + l_r}(\tan(\delta) - \tan(\delta_r)) \tag{2.21}$$

The overall equations for lateral motion become

$$\dot{X} = v\cos(\psi + \beta) \tag{2.22}$$
$$\dot{Y} = v\sin(\psi + \beta) \tag{2.23}$$
$$\dot{\psi} = \frac{v\cos(\beta)}{l_f + l_r}(\tan(\delta) - \tan(\delta_r)) \tag{2.24}$$

The model described above assumes that there is an input v, δ and δ_r. This is not the case in this thesis, as it is assumed that the orientation of the rear wheel is always assumed to be zero, and because of low speeds, the slip angle is also assumed to be zero. With these assumptions, (2.22) - (2.24) become

$$\dot{X} = v\cos(\psi) \tag{2.25}$$
$$\dot{Y} = v\sin(\psi) \tag{2.26}$$
$$\dot{\psi} = \frac{v}{l_f + l_r}\tan(\delta) \tag{2.27}$$

2.2 Model summary

The model describing the car was first introduced in section 2.1. Here the non-linear model was introduced, and the formulation of the different states was shown. To summarize, the non-linear model is expressed as

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \ddot{X} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \frac{v}{l_f + l_r}\tan(\delta) \\ \frac{V \cdot K \cdot throttle \cdot r}{v \cdot G \cdot m + \epsilon} - \frac{1}{2}\rho C_d A_f v^2 \end{bmatrix}. \tag{2.28}$$

One problem with this formulation is there can not be set constraints on the change in input, $\dot{\delta}$ and $\dot{throttle}$. To handle this problem, the inputs can be change to the $\dot{\delta}$ and $\dot{throttle}$ and two new states are introduced δ and $throttle$. This means that the model changes to

$$\begin{bmatrix} \dot{X} \\ \dot{Y} \\ \dot{\psi} \\ \ddot{X} \\ \dot{\delta} \\ \dot{throttle} \end{bmatrix} = \begin{bmatrix} v\cos(\psi) \\ v\sin(\psi) \\ \frac{v}{l_f+l_r}\tan(\delta) \\ \frac{V\cdot K\cdot throttle\cdot r}{v\cdot G\cdot m+\epsilon} - \frac{1}{2}\rho C_d A_f v^2 \\ \dot{\delta} \\ \dot{throttle} \end{bmatrix}. \tag{2.29}$$

With this there can now be set limitations on both $\dot{\delta}$ and $\dot{throttle}$. This is explained further in Section 3.1.2.

Chapter 3

Control System Design

In this chapter an introduction to what Model Predictive Control is presented in Section 3.1.2. The computation of the energy efficiency term is presented in Section 3.8. The MPC problem formulation used in this thesis is described in Section 3.3, where the objective function, constraints and tuning parameters are persented.

3.1 Model Predictive Control

Model predictive control is a control technique that is used for multi-variable control problems [18]. The first usage of MPC can be traced back to the 1960s, but it had its first breakthrough in the 1980s [6]. The concept of how the MPC works is that the MPC predicts future output values based on a process model. A prediction is made at each time step. Figure 3.1 shows an example of how a single input MPC works. At sampling instant k, the MPC calculates a set of future inputs such that future outputs $\hat{y}$ reach the desired target.

3.1.1 The Advantages of MPC

MPC is a type of control that is of open nature. This means that it applies to many different kinds of research fields such as process industry, robots, clinical anesthesia, and many more [19]. Some of the advantages of using MPC to control an autonomous vehicle are that it can handle constraints, and it is easy to change control objectives. This makes it favorable where an increase in complexity of either the objective function or prediction model is needed without changing the entire system. This is useful in student competitions such as the Shell Eco-marathon, where existing teams want to develop previous team's work without changing the entire system. The MPC is also practical when unpredictable events happen. Since the MPC predicts future outputs at every time-step, if the vehicle's position were to change due to unknown circumstances suddenly, the MPC will change its predicted output such that this is handled.

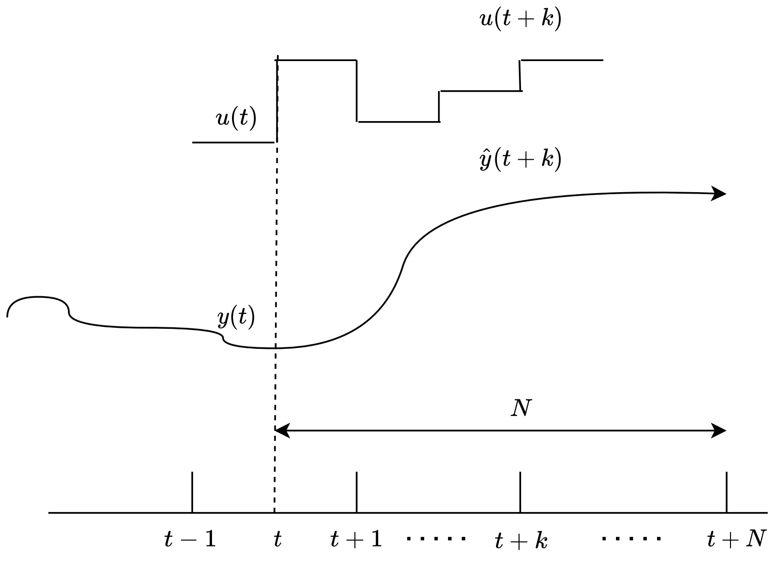


Figure 3.1: MPC strategy

3.1.2 General Formulation

There exist many different types of MPC algorithms, but what they all have common elements. These elements are:

- Prediction model
- Objective function
- Constraints

The MPC can be represented as an optimization problem formulated in

$$\min \; J(x,u) \tag{3.1a}$$
$$\text{subject to } c_i(x) = 0, \quad i \in \mathcal{E} \tag{3.1b}$$
$$c_i(x) \geq 0, \quad i \in \mathcal{I} \tag{3.1c}$$

where $J(x,u)$ is the optimization problem and $c_i(x)$ are the constraints.

Prediction Model

The purpose of the prediction model is to capture the necessary mechanisms of the process such that predictions of future outputs can be calculated. It is important to use a prediction model that captures the dynamics of the system and at the same time is able to be computed in real-time. The two main ways to represent prediction models are linear- and non-linear models. Linear models have the advantage of reducing computation load but may not capture the system's dynamics in non-linear territories. This thesis uses a non-linear prediction model since it is necessary to capture all dynamics, and it is later proven that the computational load is still satisfactory. The state-space representation of the non-linear prediction model is of the form

$$\dot{x}_{k+1} = g(x(k), u(k)) \tag{3.2}$$

$$y(k) = h(x(k), u(k)) \tag{3.3}$$

where x denotes the states of the system, u the control, and y the output [20]. The subscripts in (3.2) and (3.3) indicate that the state-space model is a discrete time-step model.

Objective Function

Objective functions for MPCs are often formulated as a quadratic cost function where the constraints and objective function are linear. This leads to a Quadratic Programming (QP) problem. However, in instances where the problem is a Non-Linear Problem (NLP), Sequential Quadratic Programming (SQP) has been developed to handle these types of problems [21]. The idea of SQP is to model the NLP at a given approximate solution x_k by a QP subproblem. The solution of the subproblem is then used to find a better approximation of x_{k+1}. This is iterated such that a sequence of approximations is made so that x will eventually converge to the solution x^*. The quadratic subproblems will have the form

$$(r^k)d_x + \frac{1}{2}d_x^\top B_k d_x \tag{3.4}$$

where r^k is the gradient of J and B_k is the Hessian of J. Details on how to solve QP's will not be explained in this thesis, however there exist many algorithms for solving these problems.

Constraints

Constraints are often used when there are boundaries to what the physical system can do. These constraints are implemented into the MPC such that they are taken into account when finding the optimal input. It is often desirable to constrain the input and states of the system such that they do not go under or over a specific value.

$$x^{low} \leq x \leq x^{high} \tag{3.5a}$$

$$u^{low} \leq x \leq u^{high} \tag{3.5b}$$

x is the states of the system and is bounded within a lower and upper value. The same applies for the input u. To ensure no rapid movements in the input u, it is common to also set a constraint in the change in input

$$\dot{u}^{low} \leq \Delta u \leq \dot{u}^{high}. \tag{3.6}$$

This is the reason why the state-space system presented in Section 2.2 changed its inputs from $delta$ and $throttle$ to $\dot{\delta}$ and $thr\dot{o}ttle$.

3.1.3 MPC solver

To solve the QP subproblem, there are many numerical solvers. The one used in this thesis is High-Performance Interior-Point (HPIPM). HPIPM is an open-source C-coded framework for QP problems [22]. From [22], it is shown that HPIPM is a fast solver that can solve many different types of QP problems. A fast solver is important when looking at real-time situations. If the solver uses too much computational load, the optimal trajectory calculated at a certain time-step will not align with the physical vehicle's time-step. This will result in unstable behavior when testing the MPC with real hardware.

3.2 Energy Efficiency

As mentioned in Section 1.4, one of the MPC's goals is to focus on energy efficiency. There are several ways to do this. One way is to model the powertrain and optimize for efficiency. This requires an accurate description of the powertrain, which may not be easy to acquire. Another method describes energy consumption as an equation consisting of motor torque and angular velocity. This thesis uses the latter method since this is the simplest method and will show acceptable behavior. The motor torque is expressed as

$$T_m = \frac{r}{G}(ma + mgC_r + \frac{1}{2}\rho C_d A v^2), \tag{3.7}$$

where

- T_m is the motor torque,
- r is the radius of the wheels,
- G is the total reduction ratio,
- m is the mass of the vehicle,
- g is the gravitational constant,
- C_r is the rolling resistance,
- ρ is the air density,
- C_d is the drag coefficient,
- A is the frontal area,
- v is the longitudinal velocity.

(3.7) considers only the longitudinal forces. The reason for not including lateral forces is that these forces play a significantly less role in terms of energy consumption. Using (3.7), the input energy can be calculated by integrating the input power as shown in (3.8)

$$\mathcal{E} = \int_0^{t_f} T_m \omega \, dt \tag{3.8}$$

where ω is angular velocity and t_f is the final time. By adding this term to the cost function J, the MPC will also focus on energy minimization.

3.3 MPC - Problem Formulation

There are mainly two objectives for the MPC. The first objective is to follow a given path and the second objective is to do this in an energy-efficient manner. To follow the given path, the car must ensure that the error between the vehicle's location and trajectory is zero. This is called *cross-track error*. In addition to keeping the cross-track error as zero, the angle between the car and the trajectory must also be zero. This is called *heading error*. As long as these two errors are zero, the vehicle will follow the trajectory. Since the trajectory is described as a function of x, as long as the y-value of the car and trajectory are equal at all times, the cross-track error will be zero. For the heading error to be zero, the heading angle of the trajectory and the car must be equal. The heading angle of the trajectory can be computed by taking the tangent of the derivative

$$\psi_{ref} = \arctan \frac{dy_{ref}}{dx}. \tag{3.9}$$

The equations for cross-track error and heading error become

$$e_{cte} = y_{ref} - y_{car} \tag{3.10}$$

$$e_{\psi} = \psi_{ref} - \psi_{car} \tag{3.11}$$

The second objective is to include the energy equation (3.8). If only these two objectives are used in the MPC, what will happen is that when the car's cross-track error and heading error become zero, the car will not want to move anymore. The reason for this is because of the second objective. Now that the first objective has been secured, the second objective will be energy efficiency. The MPC will then tell the car to lose all its velocity since the MPC will conclude that this is the most energy-efficient state, something that makes sense. A non-moving car will lose the least amount of energy. To counteract this, a third objective will be introduced, and that is to follow a reference velocity. By adding this as an objective, the car will always try to have a certain velocity. The cost function for the MPC thus becomes

$$J = \sum_{k=0}^{N} ||y_{ref} - y_k||_Q^2 + \sum_{k=0}^{N-1} ||u_k||_R^2 + \sum_{k=0}^{N} ||T_m \omega||_E^2 \tag{3.12}$$

where the two first sum terms regulate the cost of outputs and inputs and the last sum term regulate the energy output. The weights Q, R and E determine which of the different sum terms should be prioritized. By introducing large weight on the weights on the first two terms, the MPC focuses more on minimizing the output and input and vice versa. This means that different behaviors can be obtained by changing the weights of Q, R and E.

3.3.1 Constraints

Certain constraints have to be implemented to ensure limitations in both states and inputs. The car can not have a steering angle higher than 0.57 radians. The throttle input can also not be higher than the value one as this is seen as the maximum throttle. The derivative of throttle and the steering rotation speed is also limited to ensure smooth behavior. The physical limitations can be summarized as

$$\begin{aligned}
-0.57 \text{ rad} \leq \delta &\leq 0.57 \text{ rad} \\
-0.8 \text{ rad/s} \leq \dot{\delta} &\leq 0.8 \text{ rad/s} \\
-1 \leq throttle &\leq 1 \\
-0.33 \leq \dot{throttle} &\leq 0.33
\end{aligned}$$

3.3.2 Minimization Variables

There are not all outputs that are necessary to minimize. As mentioned earlier, the three main objectives of the MPC are to minimize the cross-track error and heading error, minimize the energy usage and follow a reference trajectory. This means that the variables that are most interesting to tune are $y = [e_{cte}\ e_{psi}\ v]^\top$ and $T_m\omega$.

3.3.3 Tuning

The weighting matrices Q, R, and E determine the performance of the MPC. Q tunes the variables e_{cte}, e_ψ and v. R tunes the inputs $\dot{\delta}$ and $\dot{throttle}$ and E tunes the energy usage $T_m\omega$. Finding the optimal weights is a trial and error process, and the whole process of finding satisfactory weights can be found in Appendix B The final tuning weight matrices values are

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 35 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \ R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \ E = 0.001 \tag{3.13}$$

With these weights, the car follows a flat line trajectory smooth and fast, as shown in Figure 3.2. There is also no sudden change in inputs. These weights represent the weights used for the Energy MPC. The Non-Energy MPC does not have any weight on E on uses only Q and R

$$Q = \begin{bmatrix} 5 & 0 & 0 \\ 0 & 35 & 0 \\ 0 & 0 & 10 \end{bmatrix}, \; R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} \tag{3.14}$$

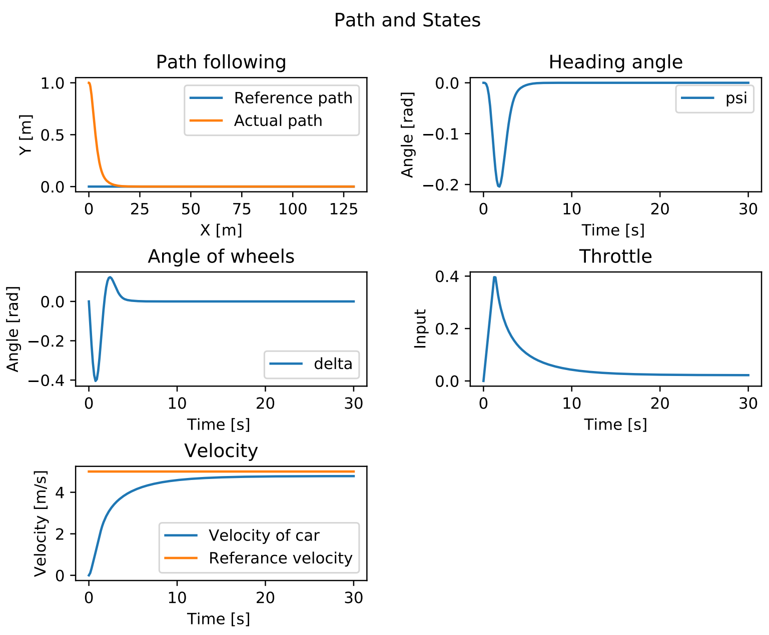


Figure 3.2: Path and state response after tuning Q, R, and E. The MPC is following a straight-line path.

Chapter 4

Implementation

When the vehicle is modelled and the MPC formulation is defined. The next step is to implement them into a simulation environment. The framework used to simulate the system is described in Section 4.1. The system will further be implemented in ROS and Gazebo. This is described in Section 4.2. To implement it into ROS and Gazebo, a path representation must be implemented. This is presented in Section 4.2.1. Lastly the values of each parameter describing the vehicle is introduced in Section 4.3.

4.1 ACADOS

The MPC is implemented using the programming language Python [23] with the combination of the framework ACADOS [24]. ACADOS is an open-source optimization framework that includes a variety of optimization methods. The library is written in C code and can be used in Python and MATLAB. Compared to other optimization frameworks, ACADOS is reasonably fast, as shown in Figure 4.1. Another advantage is that ACADOS has built-in c-code generation, making it easy to convert the Python code into c-code.

The Python code consist mainly of four different functions:

- energyBicycleModel, describing the model of the system.
- costFnc, introducing the cost function $J(x)$.
- ocpSolver, including constraints, Q, R, E values, solver method and integration types.
- main, solves the cost function at each iteration and updates the inputs and states at each iteration.

The code can be found in Appendix C. With these four functions, the optimization problem can be solved and simulated.

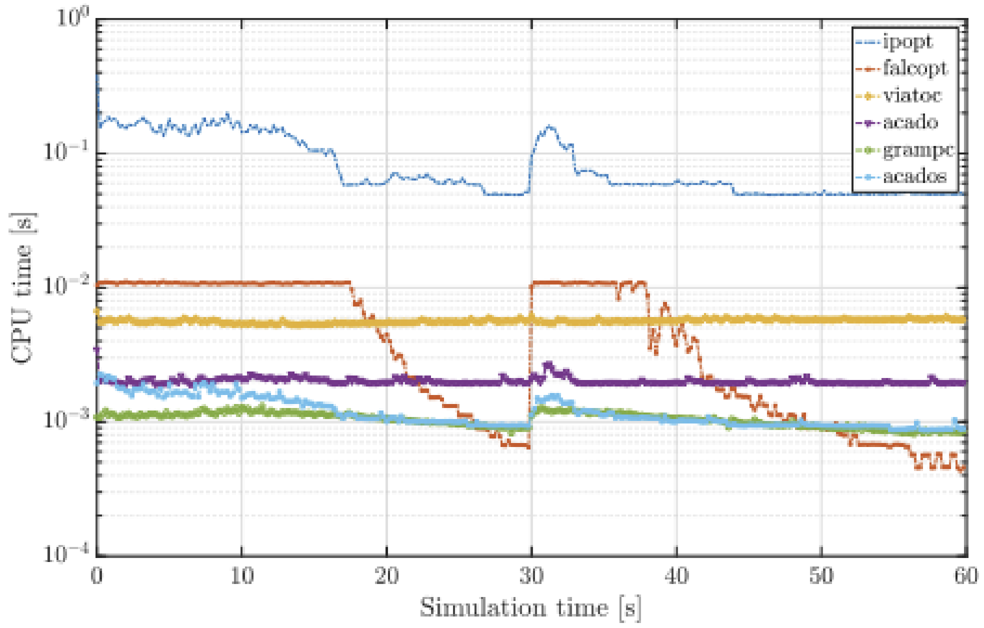


Figure 4.1: Computation time of different solvers, averaged over 10 runs (source: [24]).

4.2 Implementation with ROS and Gazebo

To connect the different subsystems described in Appendix A, Robotic Operating System (ROS) is used. ROS is an open-source software library that helps build robot applications [5]. ROS makes it easy to connect different subsystems together by publishing and subscribing to different nodes. Figure 4.2 shows an example of a flowchart of the different nodes. Here the MPC node subscribes to different nodes to receive data such as the state of the car and the path it wants to follow. The MPC then publishes the optimal velocity and steering angle, which the Gazebo subscribes to.

Gazebo is a free, open-source simulation environment where different parts of the system can be connected to see how they work in a simulated environment [25]. By adding a model of FuelFighter 5 into Gazebo and the different sensors attached to the car, a realistic simulation of the car's behavior can be simulated. This is necessary to see which of the algorithms work as intended or not. In this thesis, Gazebo is used for visual simulation to better understand how the MPC works with the car.

4.2.1 Path Representation

A third-order polynomial is used to describe this path to describe the path the vehicle wants to follow in Gazebo. Many different algorithms exist to create smooth polynomials, but this is not within the scope of this thesis. For time-saving and ease in complexity, the code used to create the third-order polynomial is inspired by `https://gist.github.com/ksjgh/4d5050d0e9afc5fdb5908734335138d0`. Using a third-order polynomial versus cubic splines is that the third-order polynomial manages to describe the trajectory at a satisfactory level, and the computation is much less demanding than cubic splines. The disadvantage with a third-order polynomial is that at sharp corners and long look-ahead distance such as that shown in Figure 4.3 the polynomial is less satisfactory. From Figure 4.3 the green line represents the actual path the car should follow, and the blue line represents the polynomial created to follow the green line. The figure shows that the polynomial fits the green line reasonably well, but the two lines stray away from each other at a further distance. However, this is not a significant problem if the green and blue lines align close to the car.

4.3 Parameters

The parameters used to describe the car can be found in Table 4.1. These values are fairly accurate as they have been gathered through data collection.

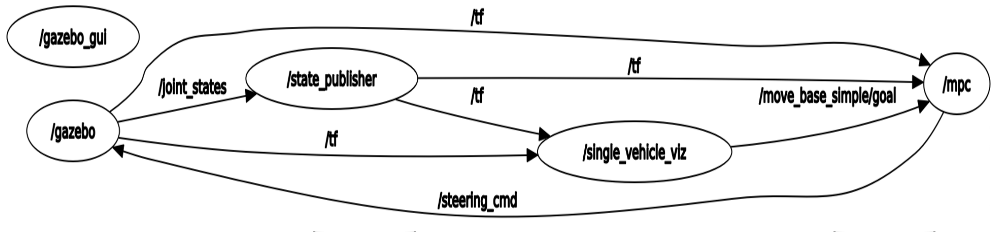


Figure 4.2: Flowchart of which nodes publish and subscribe to each other in ROS

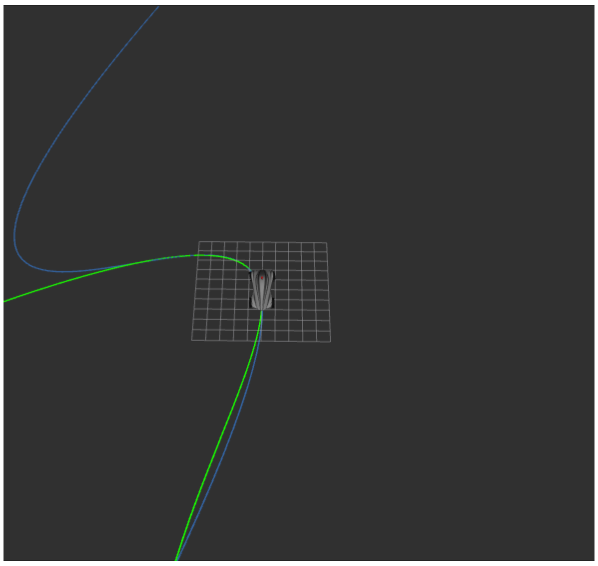


Figure 4.3: Trajectory of vehicle simulated in gazebo. The green line represents the path track and the blue line represents the third-order polynomial. Here it is shown that the polynomial does not fit the track when curves are introduced.

Parameter	Value	Description
m	70 [kg]	Mass of the car
r	0.225 [m]	Radius of the wheels
l_f	1.32 [m]	Distance between CG and front wheel
l_r	1.32 [m]	Distance between CG and rear wheel
G	0.05	Gear ratio
V	48 [V]	Voltage of batteries
C_d	0.218	Drag coefficient
ρ	1.2 [kg/m^3]	Air density
A_f	1 [m^2]	Frontal area of car
C_r	0.0015	Rolling resistance
g	9.81 [m/s^2]	Gravity constant
K	3.2	Constant

Table 4.1: Vehicle parameters

Chapter 5

Results

This chapter presents the results that are obtained by looking at three different categories,

- performance results,
- energy efficiency, and
- computational load.

The Energy MPC and Non-Energy MPC will be compared and see how they behave in each category. The performance result looks at if the MPC's manage to follow the given path and how the states and inputs behave during the path following. The Energy efficiency category is used to see how much energy each MPC spends during the simulation. Lastly, computational load is compared. The algorithm must solve the problem fast when using real hardware since slow computation can lead to stability issues.

The Energy MPC will be simulated in Gazebo, and a more advanced scenario will be simulated. This is for both visualizations and to see how the Energy MPC's performance when combined with ROS and Gazebo.

5.1 Performance

Two scenarios are conducted for testing the performance of between the two MPC's,

- straight-line, and
- sinus wave function.

These two experiments will validate the effectiveness of the MPC.

5.1.1 Straight-Line Path Following

The experiment for straight-line driving is experimented with with an initial cross-track error, $e_{cte} = 1\text{m}$. The reference velocity the car wants to follow is $v = 5\text{m/s}$.

Straight-line path using Energy MPC

The path of the car using the energy MPC is shown in Figure 5.1. The states of car are shown in Figure 5.2 and the inputs of the car are shown in Figure 5.3.

Straight-line path using Non-Energy MPC

The path of the car using the non-energy MPC is shown in Figure 5.4. The states of the car are shown in Figure 5.5 and the inputs are shown in Figure 5.6.

5.1.2 Sinus Wave Path following

The sinus wave path is given by the function $y(x) = 10 \sin \frac{x}{10}$. The car starts with a $e_{cte} = 0$. The reference velocity the MPC's track is $v_{ref} = 5\text{m/s}$.

Sinues wave path using Energy MPC

The path of the car is shown in Figure 5.7. The states of the car is shown in Figure 5.8, and the inputs are shown in Figure 5.9.

Sinus wave path using Non-Energy MPC

The path of the car is shown in Figure 5.10. The states of the car is shown in Figure 5.11, and the inputs are shown in Figure 5.12.

5.2 Energy Efficiency

To measure the energy efficiency in both the Energy MPC and the Non-Energy MPC, the energy at each time-step is calculated using (3.8). The sum is added. Since the two different MPC's do not end at the same position, The two MPC's will be compared after the car has driven 50 meters in the longitudinal direction. The trajectories that will be used are the same ones that have been used earlier. Figure 5.13 shows two plots, one with the energy usage of the Energy MPC and another with the Non-Energy MPC when following a straight-line trajectory. Figure 5.14 shows Energy MPC and the Non-Energy MPC when following a sinus wave trajectory. A summary of the total energy used can be found in Table 5.1

Path	Controller	Energy Usage [Joule]
Straight-line	Energy MPC	940
Straight-line	Non-Energy MPC	1079
Sinus wave	Energy MPC	991
Sinus wave	Non-Energy MPC	1124

Table 5.1: Energy consumption of the two MPC's when used at both scenarios

5.3 Computational Load

The computational load is calculated using the Python library *time*. The computational load is only calculated when the MPC problem is solved.

Code listing 5.1: Calculating computational load

```
start = time.time()
solver_status = ocp_solver.solve() # Solve the ocp at current iteration
t = time.time() - start
```

The load is calculated for both the Energy MPC and Non-Energy MPC after 100 iterations. The computational load for both straight-line trajectory and sinus wave trajectory are used and is shown in Figure 5.15 and Figure 5.16. A table showing the average time used for each MPC at each path is shown in Table 5.2

Path	Controller	Average time [ms]
Straight-line	Energy MPC	0.89
Straight-line	Non-Energy MPC	0.49
Sinus wave	Energy MPC	0.99
Sinus wave	Non-Energy MPC	0.82

Table 5.2: Average computational time the different MPC's use at both scenarios

5.4 Performance in Gazebo

The path the Energy MPC wants to follow is shown in Figure 5.17. Since Gazebo is a visualization tool, a video better explains how the MPC behaves in Gazebo. The link to the video is `https://www.youtube.com/watch?v=A630Ou0kKXw&ab_channel=ErikFagerli`. The video shows two simulations of the Energy MPC, where the first one shows the simulation with a prediction horizon $N = 20$ and the other shows with a prediction horizon $N = 40$. The green line represents the track the vehicle wants to follow, the blue line is the third-order polynomial, and the red line is the trajectory of the MPC. Only the first two turns are shown in the videos since that is all that is necessary to determine the behavior of the MPC.

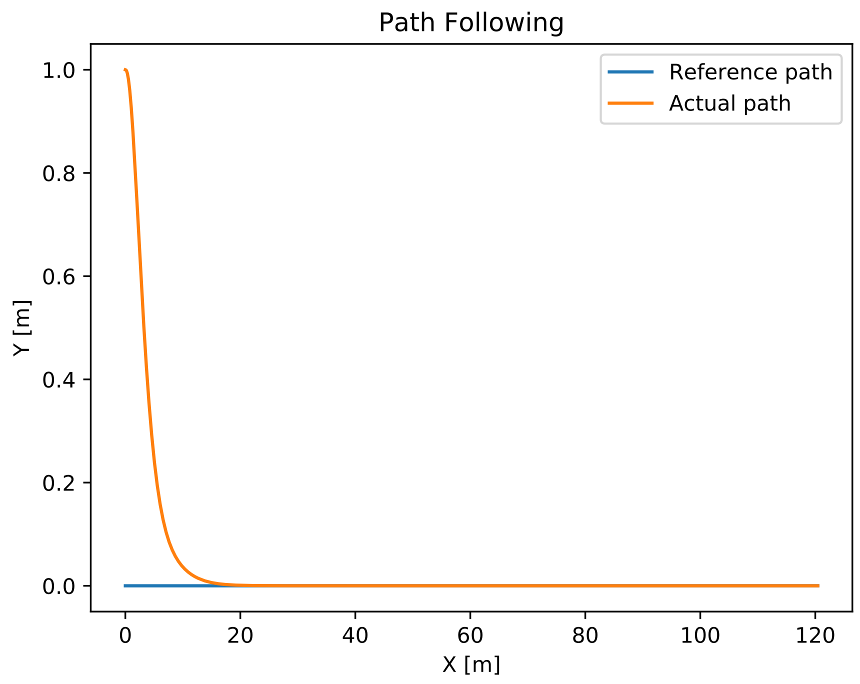


Figure 5.1: Path following of straight-line using Energy MPC

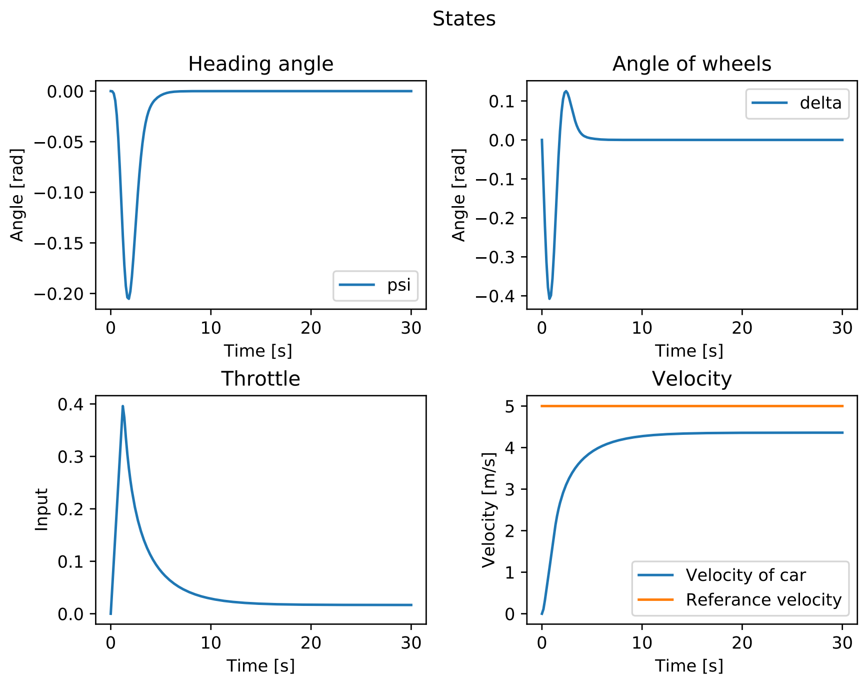


Figure 5.2: States of car when driving a straight-line using Energy MPC

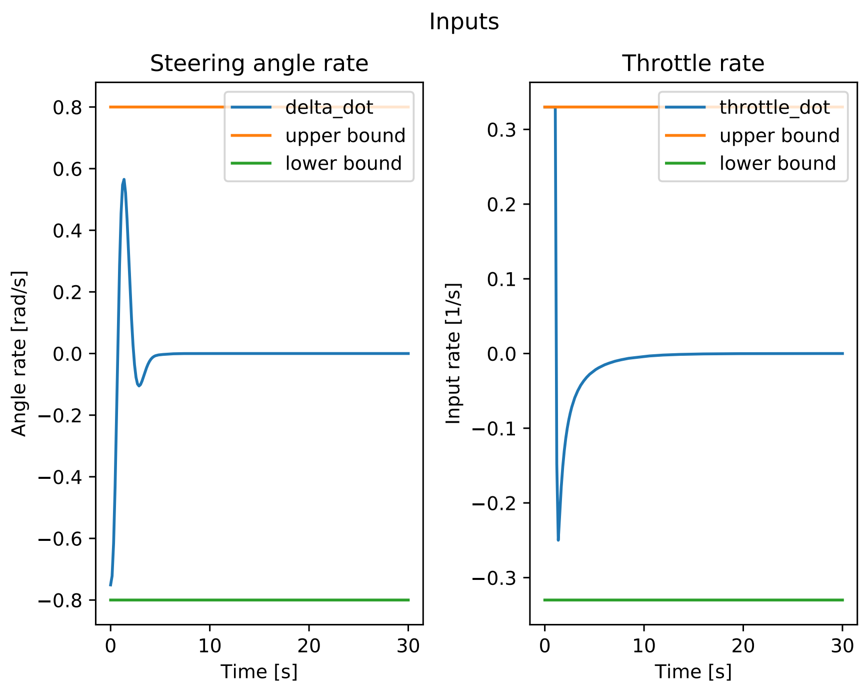


Figure 5.3: Inputs of car when driving straight-line and using Energy MPC

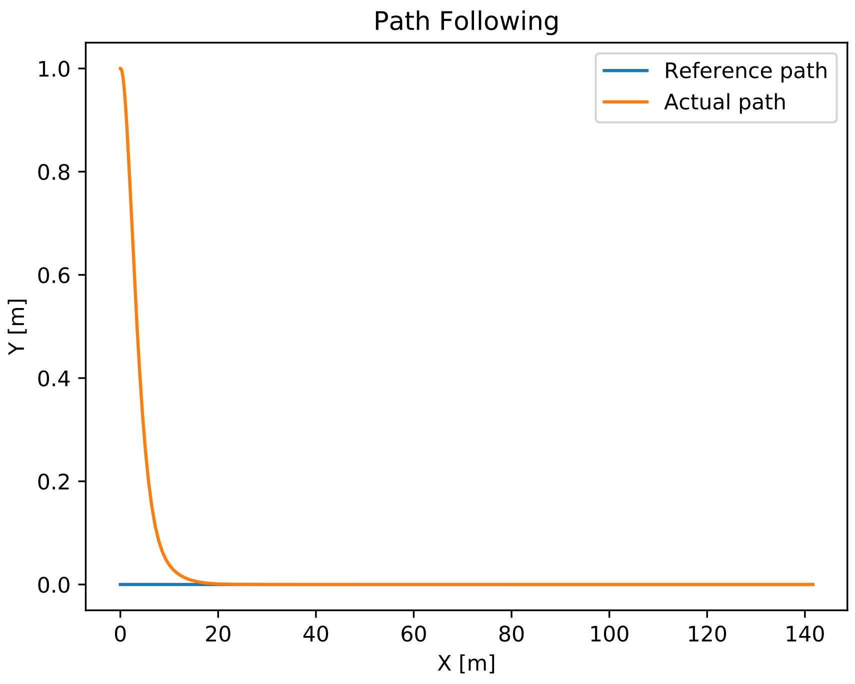


Figure 5.4: Path following of straight-line using Non-Energy MPC

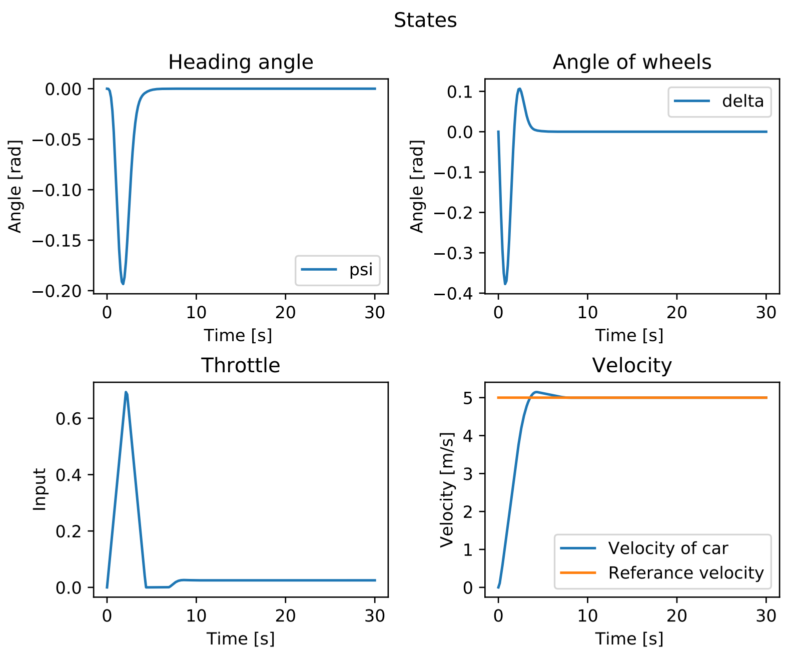


Figure 5.5: States of car when driving a straight-line using Non-Energy MPC

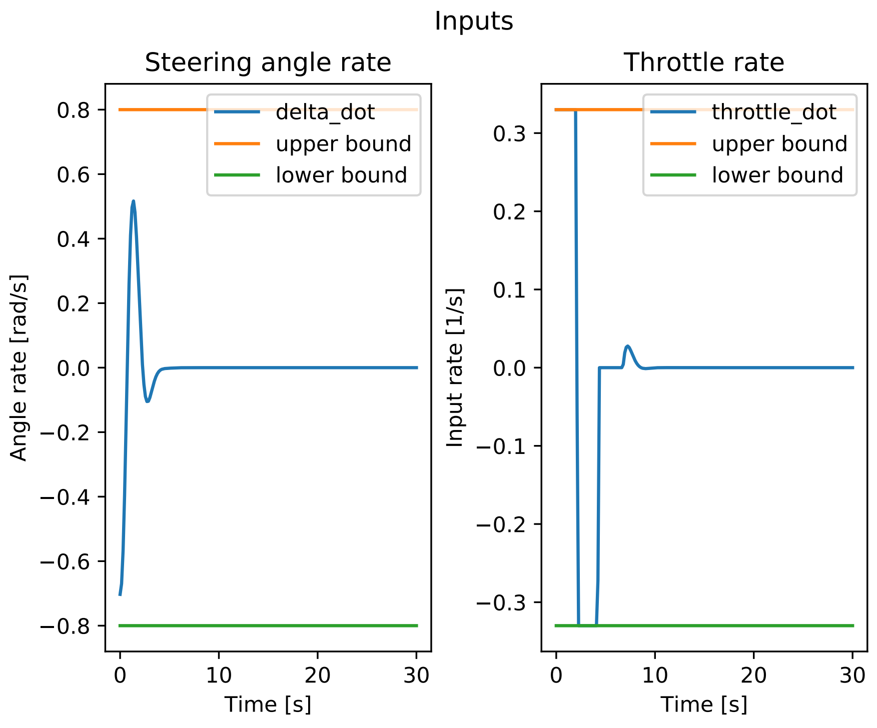


Figure 5.6: Inputs of car when driving straight-line and using Non-Energy MPC

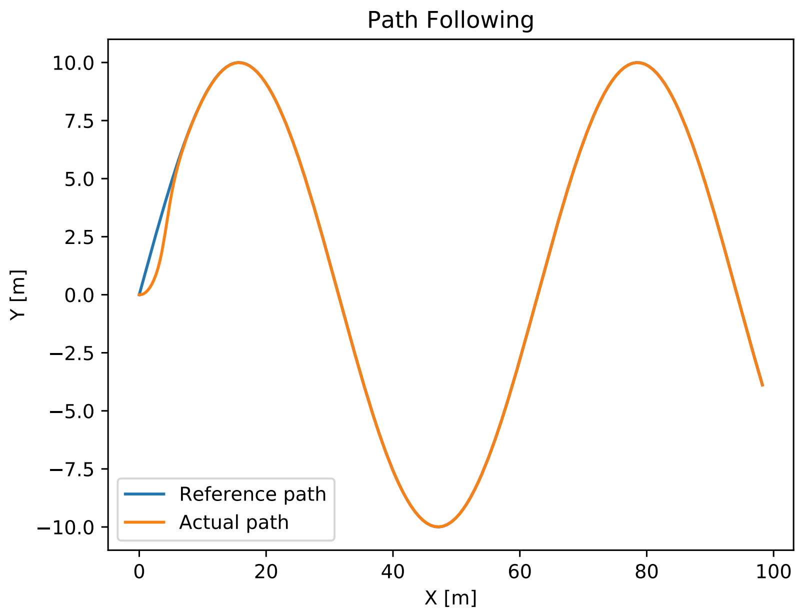


Figure 5.7: Path following of sinus wave using Energy MPC

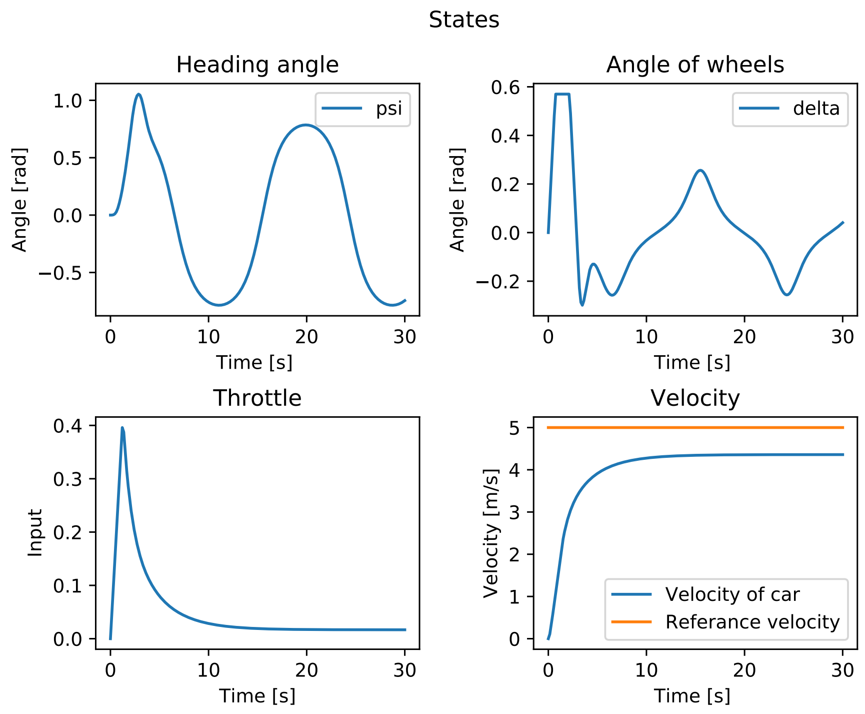


Figure 5.8: States of the car when following a sinus wave using Energy MPC

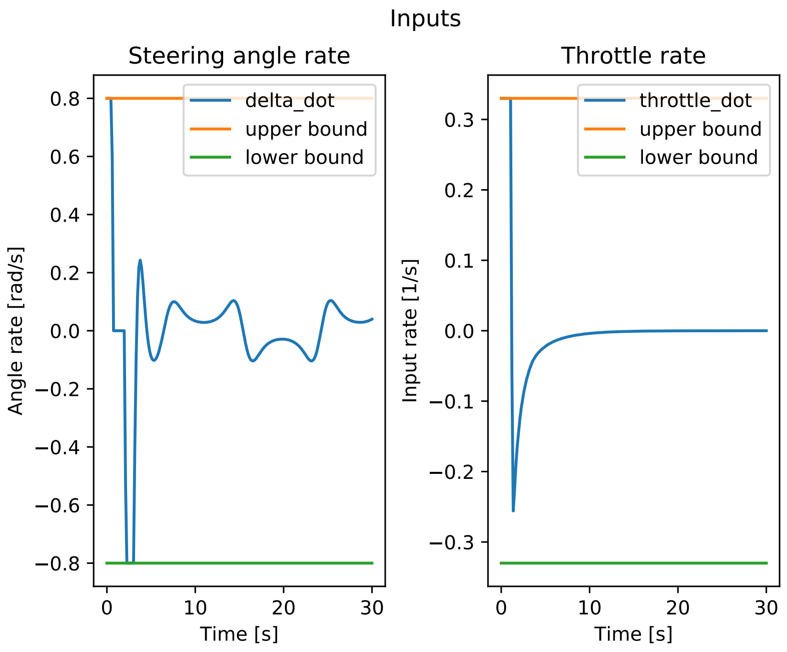


Figure 5.9: Inputs of the car when following a sinus wave using Energy MPC

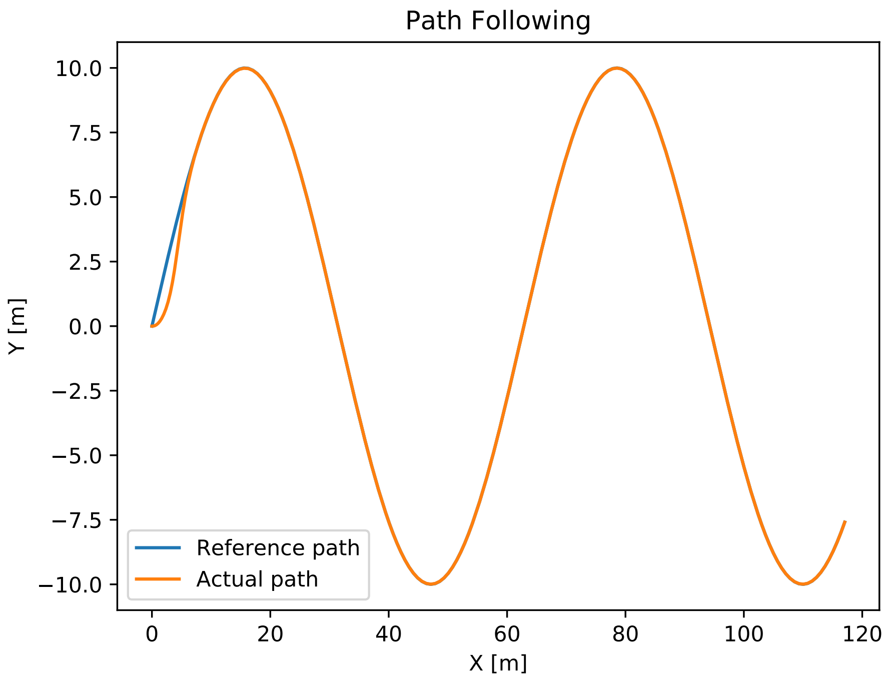


Figure 5.10: Trajectory following of sinus wave using non-energy MPC

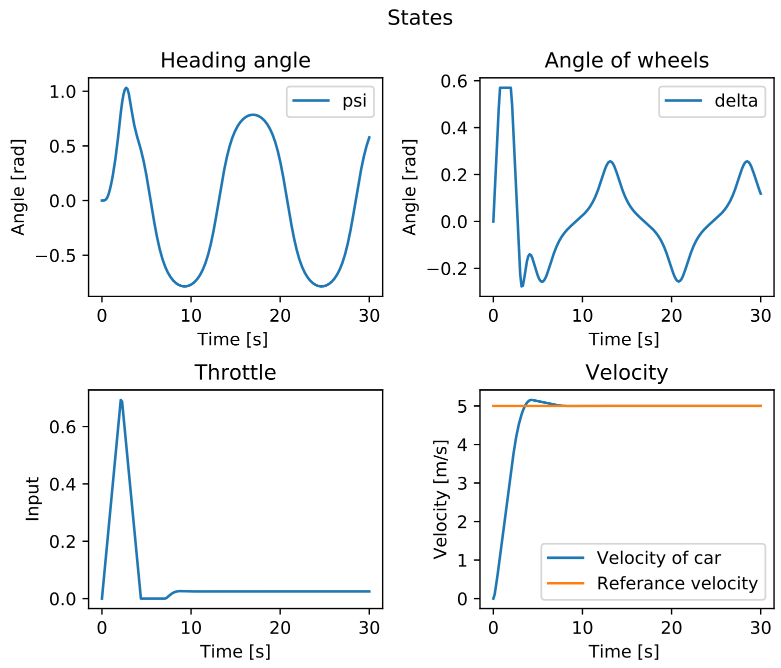


Figure 5.11: States of the car when following a sinus wave using Non-Energy MPC

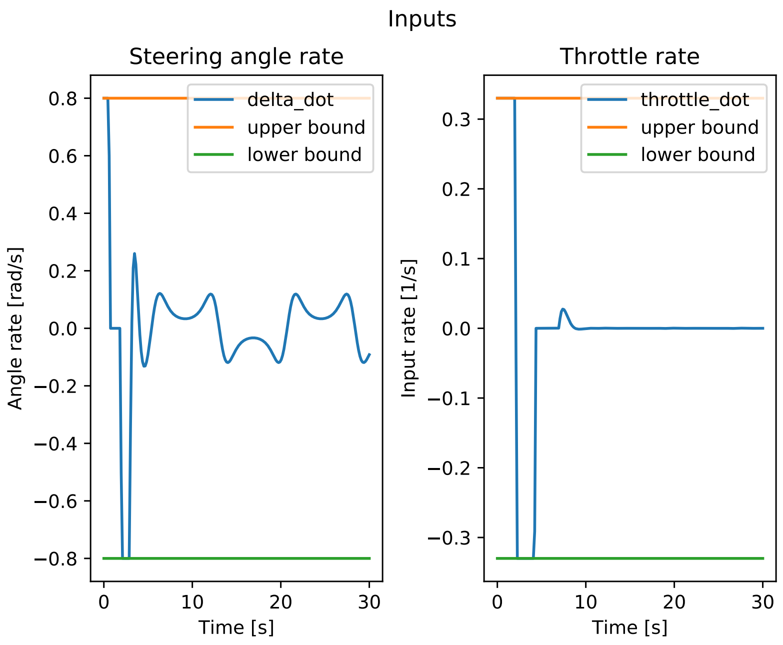
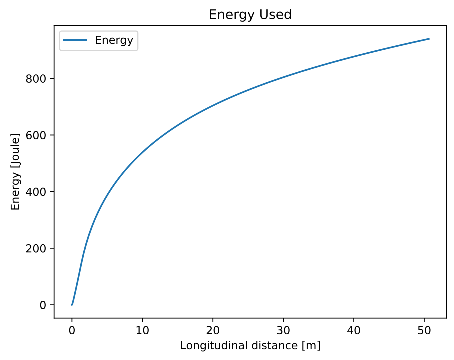
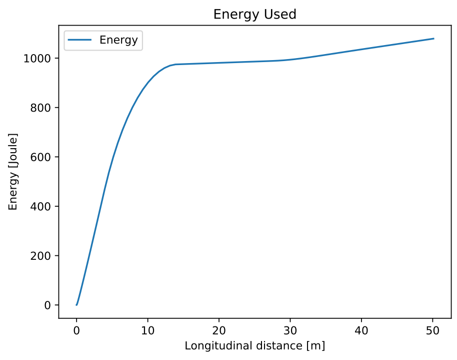


Figure 5.12: Inputs of the car when following a sinus wave using Non-Energy MPC

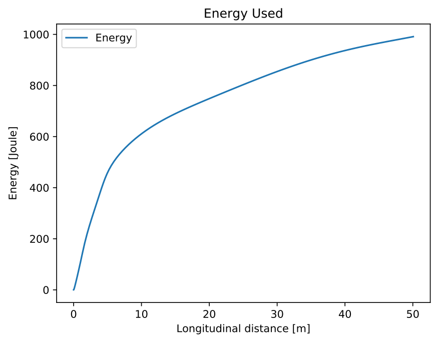


(a) Energy MPC

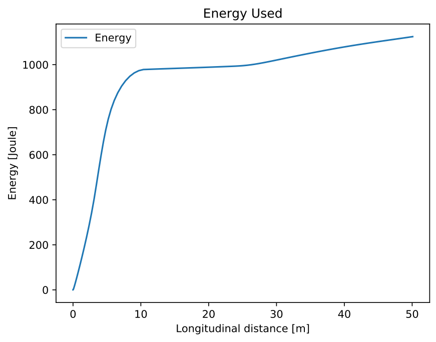


(b) Non-Energy MPC

Figure 5.13: Energy usage between Energy MPC and Non-Energy MPC when following straight-line trajectory

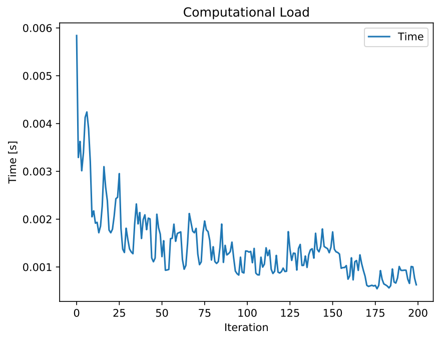


(a) Energy MPC

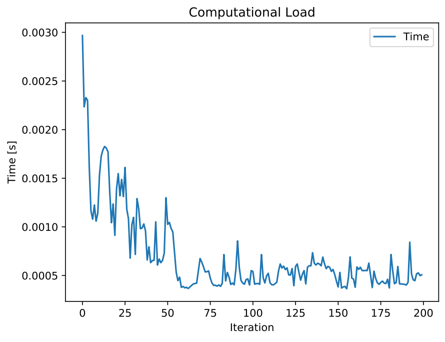


(b) Non-Energy MPC

Figure 5.14: Energy usage between Energy MPC and Non-Energy MPC when following a sinus wave trajectory

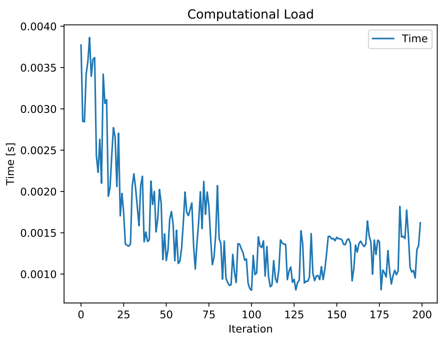


(a) Computational load of Energy MPC

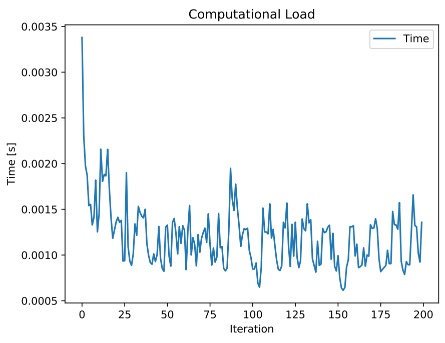


(b) Computational load of Non-Energy MPC

Figure 5.15: Computational load of both Energy MPC and Non-Energy MPC when following straight-line trajectory



(a) Computational load of Energy MPC



(b) Computational load of Non-Energy MPC

Figure 5.16: Computational load of Energy MPC and Non-Energy MPC when following a sinus wave trajectory

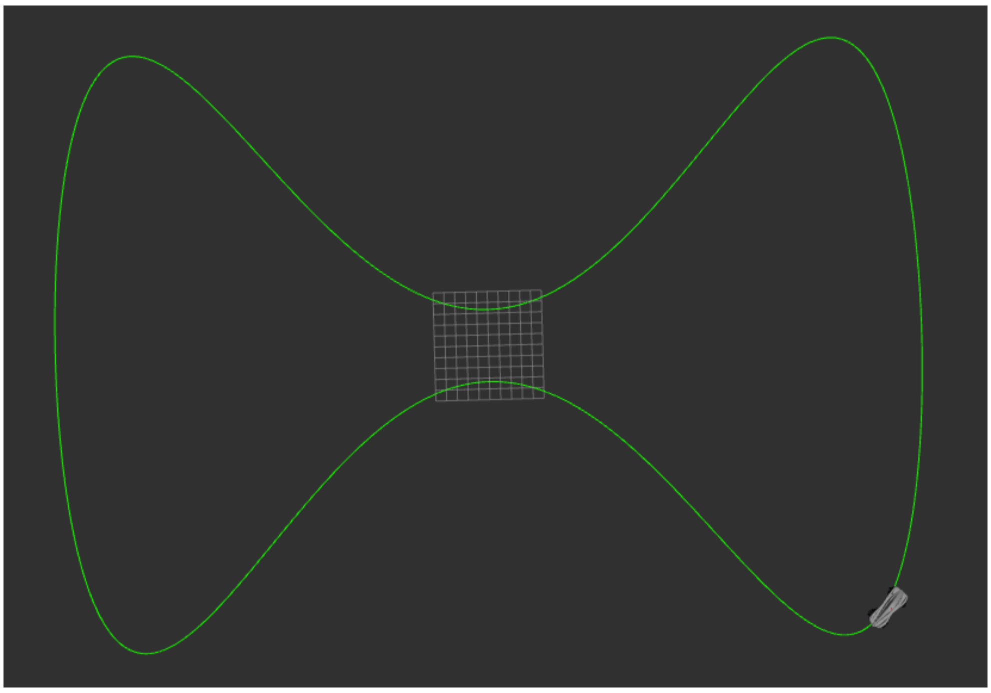


Figure 5.17: Path the car wants to follow in when simulated in Gazebo

Chapter 6

Discussion

The results presented in Chapter 5 will be discussed. The performance of both Energy MPC and the Non-Energy MPC will be discussed in Section 6.1. The amount of energy used by each MPC will be discussed in Section 6.2, the computational load is discussed in Section 6.3, and the performance of the Energy MPC in Gazebo is discussed in Section 6.4.

6.1 Performance

6.1.1 Straight-line

The performance of the Energy MPC is shown in Figure 5.1. The MPC has a $e_{cte} = 1\text{m}$ and $e_{\psi} = 0$. The figure shows that the MPC converges to the reference path fast. The states of the car are shown in Figure 5.2. There is no sudden movement in the states, and the velocity reached is 4.8m/s. The Energy MPC does not manage to climb to 5m/s because of the energy term. The energy term wants to keep the velocity low since this is more favorable for reducing energy usage. Figure 5.3 show the inputs of the system. Here it can be observed that inputs are within the constraints as expected.

The performance of the Non-Energy MPC has similar results as the Energy MPC in terms of performance. The main difference is that the Non-Energy MPC follows the reference velocity of 5m/s. In Figure 5.5 it is shown that the velocity overshoots a little before stabilizing at 5m/s. This is most likely due to the tuning of the Q and R. Since the velocity is higher, the distance traveled will also be greater when following the path; something can be seen when comparing Figure 5.4 to Figure 5.1. The inputs of the Non-Energy MPC are also within the constraints, but the throttle rate is more aggressive than the Energy MPC.

6.1.2 Sinus Wave

For the path following of the sinus wave function $f(x) = 10\sin(\frac{x}{10})$ both the Energy MPC and Non-Energy MPC start with a $e_{cte} = 0$ and $e_{\psi} = 0$. Figure 5.7 shows the path following the Energy MPC. The MPC deviates from the path initially but converges to the path quickly. The velocity does not manage to converge to the reference velocity $v_{ref} = 5\text{m/s}$ as shown in Figure 5.8. The reason for this is the same as the straight-line experiment. Figure 5.9 shows the inputs of the system, and the steering rate and throttle rate only show sudden change when it is trying to converge to the path trajectory. The control inputs are smooth when the MPC has aligned itself with the path.

The Non-Energy MPC has a similar behavior as the Energy MPC. The main difference is the input of the vehicle, as shown in Figure 5.12. The control inputs are more aggressive. The reason for this is that the Non-Energy MPC prioritizes reaching v_{ref}. This leads to the inputs being more aggressive.

6.2 Energy Usage

In Section 5.2, the energy usage of both the Energy MPC and Non-Energy MPC were calculated. The energy usage was calculated for both the straight-line and sinus wave experiments. Figure 5.13 and 5.14 show the energy used with both the Energy MPC and the Non-Energy MPC after driving 50 meters. The total amount of energy used after 50 meters can be found in Table 5.1. The table shows that the Energy MPC uses less energy, approximately 139 Joule less when following a straight-line path and 133 Joule less when following a sinus wave. The Energy MPC uses 13% and 12% less energy. Compared to the research done by [10], which achieved up to 50% less energy, it seems like the Energy MPC is worse than [10] MPC. However, in [10], the velocity of the vehicle was way lower when running energy mode compared to sport mode. Since the Energy MPC and Non-Energy MPC velocity are fairly similar, the energy output will be close. If the weight of E were to be adjusted to focus more on energy efficiency, there would be a bigger difference in energy usage between the MPCs, but it would come at the cost of a slower-moving car.

6.3 Computational Load

Computational load is important to focus on since if the MPC problem is too computational heavy, the real-life system will reach a new state before the MPC is solved. This will result in unstable behavior and is undesirable. The computational load of both the Energy MPC and Non-Energy MPC can be found in Figure 5.15 and 5.16. Figure 5.15 shows the amount of time each iteration uses to solve the MPC problem when the car follows a straight-line path, while Figure 5.16 shows the computational load when the car follows a sinus wave. The computational load of both experiments is summarized in Table 5.2. From the figures and the table, it is seen that the Non-Energy MPC uses less time to solve the problem compared to the Energy MPC. This makes sense since the Energy MPC has to consider the energy term. This term is nonlinear and thus leads to more computation.

Nevertheless, both the MPCs solve the problem at a satisfying rate.

6.4 Performance in Gazebo

From the video presented in Section 5.4, the Energy MPC behaves satisfactorily when the horizon is $N = 20$. The car does not deviate from its path, and no sharp movements are made. However, when the horizon is changed to $N = 40$, the MPC does not manage to follow the given path. In the first turn, it manages to follow the path, but in the second turn, the MPC does not manage to solve the problem and destabilizes. This happens because the MPC does not manage to find the optimal path to follow. It is most likely due to the sharp turn. When the car enters a sharp turn, the generated third-order polynomial starts to vary a lot and becomes a complex trajectory to follow. When the car first destabilizes, it becomes difficult to correct itself. This means that when implementing the algorithm into hardware, N needs to also be tuned so that the vehicle does not drift away from the path.

Chapter 7

Conclusion and Future Work

Throughout this thesis, a controller for both longitudinal and lateral path following has been designed and implemented. The multi-objective MPC focused on path following, energy efficiency and following a referance velocity. Energy efficiency was evaluated by comparing it to another MPC that did not focus on energy usage. The MPC was implemented in Python using the framework ACADOS and the solver HPIPM. The algorithm was then later extended to work together with ROS and Gazebo. The conclusion of the thesis is presented in Section 7.1. Further work that can be done to improve the MPC is presented in Section 7.2.

7.1 Conclusion

This thesis has tested an Energy MPC and Non-Energy MPC in different scenarios. To test the MPC's, a vehicle model had to be implemented, then the MPC's cost function, and constraints had to be decided. The two MPC's have been tested in performance, energy usage, and computational load. The Energy MPC performs well when following a straight-line path; even with an initial cross-track error, the MPC converges quickly to the given reference path. When following a sinus wave, the Energy MPC also converges quickly to the reference path and follows the path through the whole simulation. The Non-Energy MPC shows similar results as the Energy MPC, but with more aggressive inputs.

Calculating the energy usage of both the Energy MPC and the Non-Energy MPC showed that the Energy MPC used less energy in both scenarios. This is due to the energy cost term. The downside is that the MPC has a more challenging time following the velocity trajectory.

The computational load showed that the Non-Energy MPC used less time to solve the MPC problem. This is expected since the Non-Energy MPC did not include the energy term, thus making the MPC a less computational-heavy problem. Even though Energy MPC had a higher computational load, it is still within acceptable execution time.

Lastly, the Energy MPC was implemented with ROS and Gazebo and simulated in a more realistic scenario. The results showed that Energy MPC followed the given path without deviating. However, with an increased prediction horizon N, the MPC had difficulties following the path at sharp turns and ended with the MPC not being able to solve the problem.

Depending on what FuelFighter wants to prioritize, both MPC's work. If energy efficiency wants to be prioritized, the Energy MPC works well. However, if a certain velocity wants to be achieved, the Non-Energy MPC suits better. Both MPC's show satisfactory performance levels in terms of path following.

7.2 Future Work

Even though both MPC's perform well in this thesis, it does not mean they will perform as intended in a more realistic environment. Many features can be implemented to improve the performance further, and some are mentioned in the following section.

7.2.1 Additional Features

Path-Parametric Reformulation

By changing the vehicle model so that it is transformed from time-dependent vehicle kinematics to track-dependent (spatial) kinematics [26], new constraints can be introduced, such as road bounds. This makes it so that the vehicle can more easily cut corners if needed, thus using less time to reach the destination and saving energy.

Linearization

The vehicle model is, as of now, a non-linear model. These models are more computational heavy. A way to decrease computational load is to linearize the model. This can be done by using a small-angle approximation. The downside is that the model is only valid at small angles. A way to counteract this is to use Adaptive Model Predictive Control (AMPC) [27]. The advantage of AMPC is that it linearizes the model at different operating points instead of only one. This makes it so that the linearized model is more accurate. The downside is that it is more computational heavy than just linearizing in one operating point.

System Identification

An accurate way of describing a physical model is to make a system identification. System identification is a method of building dynamic models from input-output data [28]. The disadvantage with this is that in many cases, the model becomes highly non-linear, which can drastically increase computation.

7.2.2 Testing on FuelFighter 5

After the MPC algorithm has been converted to c-code and integrated fully with the different subsystems in ROS, the MPC needs to be tested on the physical car. It is not expected that everything goes smoothly from the start, and tuning different variables such as the weighting matrices Q, R, E, and the prediction horizon N may need to be done for the car to run more smoothly.

Bibliography

[1] E. Fagerli, *Adaptive model predictive control and path planning for an autonomous car*, Specialization project, 2021.

[2] R. Bishop, 'A survey of intelligent vehicle applications worldwide,' in *Proceedings of the IEEE Intelligent Vehicles Symposium 2000 (Cat. No.00TH8511)*, 2000, pp. 25–30. DOI: 10.1109/IVS.2000.898313.

[3] S. Eco, *Shell eco-marathon: Brilliant minds coming together to help build a lower carbon world*. [Online]. Available: https://www.makethefuture.shell/en-gb/shell-eco-marathon, (accessed: 30.09.2021).

[4] FuelFighter, *History*. [Online]. Available: https://www.fuelfighter.no/history, (accessed: 30.09.2021).

[5] *Ros*, https://www.ros.org/, (accessed: 2021-12-14).

[6] M. Morari and J. H. Lee, 'Model predictive control: Past, present and future,' *Computers Chemical Engineering*, vol. 23, no. 4, pp. 667–682, 1999, ISSN: 0098-1354. DOI: https://doi.org/10.1016/S0098-1354(98)00301-9. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0098135498003019.

[7] J. Kong, M. Pfeiffer, G. Schildbach and F. Borrelli, 'Kinematic and dynamic vehicle models for autonomous driving control design,' Jun. 2015, pp. 1094–1099. DOI: 10.1109/IVS.2015.7225830.

[8] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, 'The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?' In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818. DOI: 10.1109/IVS.2017.7995816.

[9] D.-m. Wu, Y. Li, C.-q. Du, H.-t. Ding, Y. Li, X.-b. Yang and X.-y. Lu, 'Fast velocity trajectory planning and control algorithm of intelligent 4wd electric vehicle for energy saving using time-based mpc,' *IET Intelligent Transport Systems*, vol. 13, no. 1, pp. 153–159, 2019.

[10] M. A. Daoud, M. Osman, M. W. Mehrez and W. W. Melek, 'Path-following and adjustable driving behavior of autonomous vehicles using dual-objective nonlinear mpc,' in *2019 IEEE International Conference on Vehicular Electronics and Safety (ICVES)*, 2019, pp. 1–6. DOI: 10.1109/ICVES.2019.8906412.

[11] M. Zanon, 'A gauss–newton-like hessian approximation for economic nmpc,' *IEEE Transactions on Automatic Control*, vol. 66, no. 9, pp. 4206–4213, 2021. DOI: 10.1109/TAC.2020.3034868.

[12] R. Hult, M. Zanon, S. Gros and P. Falcone, 'Energy-optimal coordination of autonomous vehicles at intersections,' in *2018 European Control Conference (ECC)*, 2018, pp. 602–607. DOI: 10.23919/ECC.2018.8550367.

[13] *Autonomous urban concept competition*, https://drive.google.com/file/d/1i5ua_YHscRT5DcKQqTfDh61PQPWzxyDb/view?fbclid=IwAR17R0xsburO0Nz5zwHfosj-X9IGyt1oIk3dGhttNpvOhpErhaPaKOcPNyE, Accessed: 2022-06-05.

[14] R. Rajamani, in *Vehicle Dynamics and Control*, 2012, pp. 25–30. DOI: https://doi.org/10.1007/978-1-4614-1433-9.

[15] *Teensy usb development board*, https://www.pjrc.com/teensy/index.html, Accessed: 2022-05-13.

[16] Maxongroup, *Re 50*, Accessed: 13-05-2022.

[17] P. Polack, F. Altché, B. d'Andréa-Novel and A. de La Fortelle, 'The kinematic bicycle model: A consistent model for planning feasible trajectories for autonomous vehicles?' In *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017, pp. 812–818. DOI: 10.1109/IVS.2017.7995816.

[18] D. E. Seborg, D. A. Mellichamp and T. F. Edgar, *Process Dynamics and Control*, Third, ser. Wylie Series in Chemical Engineering. John Wiley & Sons, 2011, ISBN: 9780470646106. [Online]. Available: http://www.worldcat.org/isbn/9780470646106.

[19] C. B. E.F. Camacho, in *Model Predictive Control*, 2007. DOI: https://doi.org/10.1007/978-0-85729-398-5.

[20] H. Nijmeijer and A. van der Schaft, 'Discrete-time nonlinear control systems,' in *Nonlinear Dynamical Control Systems*. New York, NY: Springer New York, 1990, pp. 399–421, ISBN: 978-1-4757-2101-0. DOI: 10.1007/978-1-4757-2101-0_14. [Online]. Available: https://doi.org/10.1007/978-1-4757-2101-0_14.

[21] P. T. Boggs and J. W. Tolle, 'Sequential quadratic programming,' *Acta Numerica*, vol. 4, pp. 1–51, 1995.

[22] G. Frison and M. Diehl, 'Hpipm: A high-performance quadratic programming framework for model predictive control,' *IFAC-PapersOnLine*, vol. 53, no. 2, pp. 6563–6569, 2020, 21st IFAC World Congress, ISSN: 2405-8963. DOI: https://doi.org/10.1016/j.ifacol.2020.12.073. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S2405896320303293.

[23] G. Van Rossum and F. L. Drake, *Python 3 Reference Manual*. Scotts Valley, CA: CreateSpace, 2009, ISBN: 1441412697.

[24] R. Verschueren, G. Frison, D. Kouzoupis, J. Frey, N. van Duijkeren, A. Zanelli, B. Novoselnik, T. Albin, R. Quirynen and M. Diehl, 'Acados – a modular open-source framework for fast embedded optimal control,' *Mathematical Programming Computation*, Oct. 2021, ISSN: 1867-2957. [Online]. Available: https://doi.org/10.1007/s12532-021-00208-8.

[25] *Gazebo*, https://gazebosim.org/home, Accessed: 2022-05-31.

[26] J. V. Frasch, A. Gray, M. Zanon, H. J. Ferreau, S. Sager, F. Borrelli and M. Diehl, 'An auto-generated nonlinear mpc algorithm for real-time obstacle avoidance of ground vehicles,' in *2013 European Control Conference (ECC)*, 2013, pp. 4136–4141. DOI: 10.23919/ECC.2013.6669836.

[27] M. Bujarbaruah, X. Zhang, E. Tseng and F. Borrelli, 'Adaptive mpc for autonomous lane keeping,' Feb. 2018.

[28] L. Ljung, 'Perspectives on system identification,' *Annual Reviews in Control*, vol. 34, no. 1, pp. 1–12, 2010, ISSN: 1367-5788. DOI: https://doi.org/10.1016/j.arcontrol.2009.12.001. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1367578810000027.

[29] VectorNav, *Vn-300*, https://www.vectornav.com/products/detail/vn-300, Accessed: 23-05-2022.

[30] Stereolabs, *Zed 2*, https://www.stereolabs.com/zed-2/, Accessed: 23-05-2022.

[31] *Re 50*, https://www.maxongroup.com/medias/sys_master/root/8881624973342/EN-21-152.pdf, Accessed: 2022-05-25.

Appendix A

FuelFighter 5

FuelFighter 5 was created in 2019 but reconfigured in 2021. The car was originally used for the manned competitions, but it was decided it was going to be used for the autonomous competition this year. To reconfigure the car, several parts needed to be introduced to the system. Figure A.1 shows the pipeline of the autonomous system. Additional sensors needed to be added to the car, as well as actuators and new software.

A.1 Sensors

The sensors used in FuelFighter 5 are

- an Inertial Navigation System (INS),
- a Light Detection and Ranging (LiDAR) and
- a camera (ZED 2).

The INS is used to give a good estimate of the different states of the car. The INS used is a VN-300 [29]. The VN-300 is a high-performance Dual Antenna GNSS-Aided Inertial Navigation System capable of estimating with high accuracy position, velocity, and orientation.

LiDAR is a sensoring method that uses light to detect surroundings. The amount of time for the light to bounce back into the LiDAR can be used to determine the distance between different objects. LiDAR is good at detecting objects but struggles with detecting lines. To handle this, a camera is used to detect the lines. The camera used is a ZED 2 [30]. ZED 2 is a highly advanced camera used to create 3D images of the surroundings. With both LiDAR and ZED 2, both objects and lines can easily be detected.

A.2 Perception

The information gathered from the different sensors is fed into the Simultaneous Localization and Mapping (SLAM) module. This is used to create a map of the environment around the car, which depicts where the different obstacles and lines are. The obstacle and lines detected from LiDAR and ZED 2 are also used with the combination of software to create a goal or trajectory the car wants to follow.

A.3 Mid Level Controller

The MPC outputs both throttle value and steering angle. These two values are sent to a teensy [15]. The teensy takes the two values and converts them to 0-255 values. These values are then either sent to a motor controller or directly to the actuator. The motor controller used in the car is a custom-made motor controller that DNV FuelFighter has developed. It takes in the 0-255 values and converts them into amp-signals. The amp signals are then sent to the motors.

A.4 Controllers

There are three parts of the vehicle that needs to be controlled;

- The steering wheel,
- the brakes, and
- the wheels

The steering wheel is controlled using a stepper motor. This motor turns the steering wheel with precise movement. This is important since over-or under-steering may cause failure. To control the brakes, a linear actuator is used. The linear actuator pushes down the brakes depending on how much the car needs to slow down. Finally, the motors used to spin the wheels are two DC motors [31].

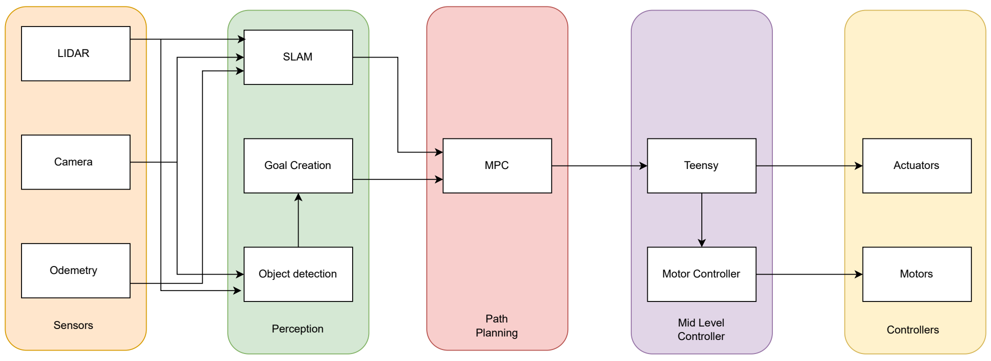


Figure A.1: Flowchart of the autonomous system

Appendix B

Tuning Weighting Matrices

The determine the weighting matrices Q, R, and E, an experiment using a straight-line path has be conducted. The initial cross-track error is set to 1m and $v_{ref} =$ 5m/s. Start by testing the weighting matrices

$$Q = \begin{bmatrix} 100 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{bmatrix}, \; R = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}, \; E = 1. \tag{B.1}$$

This results in Figure B.1. From a glance at the plot, the results look satisfactory, but the average velocity was 1.4m/s, which is way to slow. Decreasing E to 0.001 results in Figure B.2. The average velocity is now 4.4m/s, which is satisfactory. The problem now is that the system is underdamped. To fix this the weighting matrices Q needs to be tuned. This is a tidies process that follow the same procedure as when tuning E. The final values are shown in (3.13). The same values are used for the Non-Energy MPC except that $E = 0$. This gives the result shown in Figure B.3.

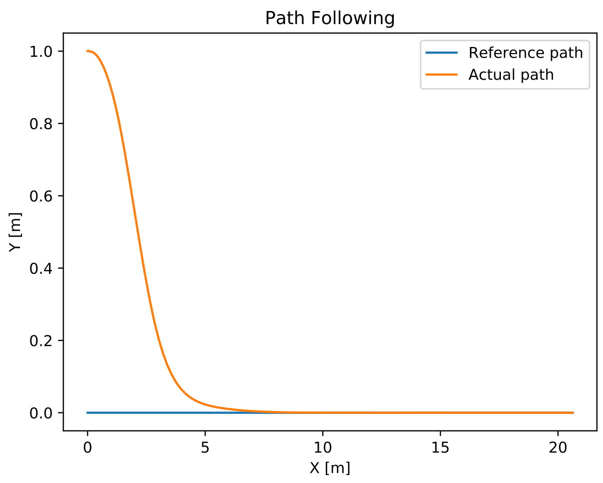


Figure B.1: Path following of Energy MPC when tuning $Q = \mathrm{diag}(100, 100, 100)$, $R = \mathrm{diag}(1, 1)$, $E = 1$

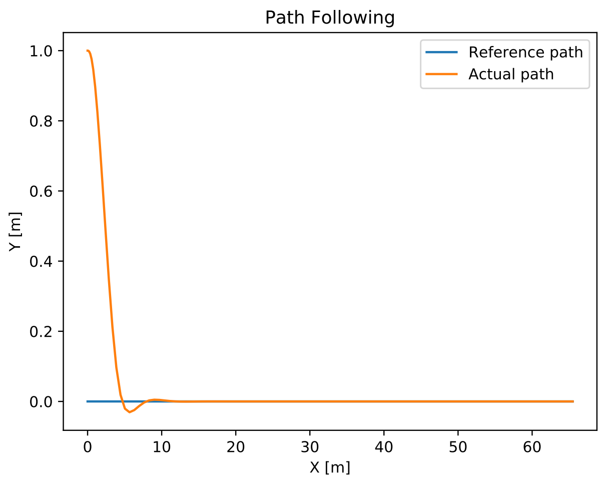


Figure B.2: Path of the car after changing $E = 0.001$

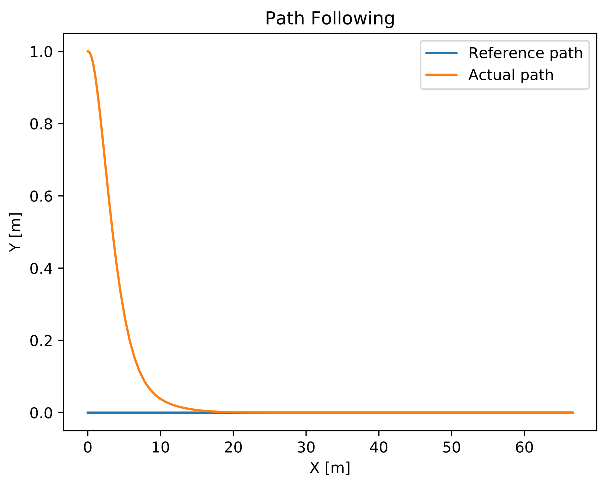


Figure B.3: Path of the car after final tuning of weights for Non-Energy MPC. $Q = \text{diag}(5, 35, 10)$, $R = \text{diag}(1, 1)$

Appendix C

Code

Code listing C.1: energyBicycleModel

```
def energyBicycleModel(params):

    modelName = "BicycleModel" # Name of model

    # Constants

    r = params["radius"] # Wheel radius [m]
    l = params["wheelbase"] # Distance between front wheel and rear wheel [m]
    G = params["gear_ratio"] # Gear ratio
    m = params["mass_car"]# Mass of car [kg]
    V = params["voltage"] # Voltage
    Cd = params["drag_coefficient"] # Drag coefficient
    rho = params["air_resistance"]# Air resistance [kg/ m  ]
    A = params["frontal_area"]# Frontal area [ m  ]

    # States

    x1 = SX.sym("x1") # X-position
    y1 = SX.sym("y1") # Y-position
    psi = SX.sym("psi") # Angle of car
    v = SX.sym("v") # Velocity of car
    delta = SX.sym("delta") # Steering angle
    throttle = SX.sym("throttle") # Throttle


    x = vertcat(x1, y1, psi, v, delta, throttle)

    # Input

    delta_dot = SX.sym("delta_dot") # Steering angle derivative
    throttle_dot = SX.sym("throttle_dot") # Throttle derivative

    u = vertcat(delta_dot, throttle_dot)

    x1Dot = SX.sym("x1_dot")
    y1Dot = SX.sym("y1_dot")
    psiDot = SX.sym("psi_dot")
    a = SX.sym("a")
    delta_dot_state = SX.sym("delta_dot_state")
    throttle_dot_state = SX.sym("throttle_dot_state")
    xDot = vertcat(x1Dot, y1Dot, psiDot, a, delta_dot_state, throttle_dot_state)

    f_expl = vertcat(
        v * (cos(psi)), # x_dot
        v * (sin(psi)), # y_dot
        v * tan(delta)/l, # psi_dot
        V * 3.2 * throttle * r / (v * G * m + 1) - (1/2*(rho*Cd*A*(v)**2) / m), # a
        delta_dot,
        throttle_dot
    )
    #Parameters used in describing third-order polynomial
    p = vertcat(SX.sym("coeff_0"), SX.sym("coeff_1"), SX.sym("coeff_2"),
                SX.sym("coeff_3"))

    f_impl = xDot - f_expl
    model = AcadosModel() # Class containing all information of the model

    # Add the respective states, inputs and so on to the different model
    model.name = modelName
    model.f_expl_expr = f_expl
    model.f_impl_expr = f_impl
    model.xdot = xDot
    model.x = x
    model.u = u
    model.p = p

    return model
```

Code listing C.2: costFnc

```
def costFnc(model):

    # Opem yaml file with different constants
    with open("../params/mpc.yaml", "r") as paramFile:
        params = yaml.safe_load(paramFile)

    r = params["radius"] # Radius of wheel [m]
    m = params["mass_car"] # Mass of vehicle [kg]
    Cr = params["rolling_resistance"] # Rolling resistance
    Cd = params["drag_coefficient"] # Drag coefficient
    rho = params["air_resistance"] # Air resistance [kg/m  ]
    A = params["frontal_area"] # Frontal area [m  ]
    g = params["gravity_constant"]# Gravity constant [m/s  ]
    G = params["total_reduction_ratio"]# Total reduction ratio

    # Define what is going to be in cost function
    x1 = model.x[0]
    y1 = model.x[1]
    psi = model.x[2]
    v = model.x[3]
    delta = model.x[4]
    throttle = model.x[5]

    delta_dot = model.u[0]
    throttle_dot = model.u[1]

    a = model.f_expl_expr[3]

    coeffs = model.p
    Tm = r/G * (m*a + m*g*Cr + 1/2*rho*Cd*A*(v)**2) # Torque

    energy = Tm * v/r # Energy equation

    # Different paths to test

    # Sinusfunction
    #yPath = 10*sin(x1/30)
    #pathYaw = atan(1/3*cos(x1/30))

    # Third-order polynomial
    #yPath =  coeffs[3]*x1**3 + coeffs[2]*x1**2 + coeffs[1]*x1 + coeffs[0]
    #pathYaw = atan(3*coeffs[3]*x1*x1 + 2*coeffs[2]*x1 + coeffs[1])

    # Straight line
    yPath = 0
    pathYaw = 0

    # Cross-track error and heading error
    epsi = psi - pathYaw
    cte = yPath - y1

    # What variables we want to minimize
    return vertcat(cte, epsi, v, delta, throttle, delta_dot, throttle_dot, energy)
```

Code listing C.3: ocpSolver

```
def ocpSolver():
    # Open params file
    with open("../params/mpc.yaml", "r") as paramFile:
        params = yaml.safe_load(paramFile)

    # Create render arguments
    ocp = AcadosOcp()

    # export model
    ocp.model = energyBicycleModel.energyBicycleModel(params)

    # Set dimensions
    nx = ocp.model.x.size()[0]
    nu = ocp.model.u.size()[0]
    ny = nx + nu

    N = params["mpc_N"] # Number of steps
    dt = params["mpc_dt"] # Time steps
    Tf = N*dt

    ocp.dims.N = N

    # Constraints
    deltaMax = params["max_steering_angle"]
    deltaDotMax = params["max_steering_rotation_speed"]

    throttleMin = 0
    throttleMax = params["throttle_max"]
    throttleDotMax = params["throttle_dot_max"]

    ocp.constraints.constr_type = "BGH"
    ocp.constraints.lbx = np.array([-deltaMax, throttleMin]) # Lower bound on state
    ocp.constraints.ubx = np.array([deltaMax, throttleMax]) # Upper bound on state
    ocp.constraints.idxbx = np.array([4, 5])
    ocp.constraints.lbu = np.array([-deltaDotMax, -throttleDotMax]) # Lower bound on input
    ocp.constraints.ubu = np.array([deltaDotMax, throttleDotMax]) # Upper bound on input
    ocp.constraints.idxbu = np.array([0, 1])

    x0 = np.array([0, 0, 0, 0, 0, 0])
    ocp.constraints.x0 = x0

    param = np.array([0, 0, 0, 0])
    ocp.parameter_values = param

    # Cost

    ocp.cost.cost_type = "NONLINEAR_LS"
    ocp.cost.yref = np.array([0, 0, 5, 0, 0, 0, 0, 0]) # Reference the different
    # variables in cost function should follow
    ocp.model.cost_y_expr = energyBicycleModel.costFnc(ocp.model)
    ocp.cost.W = np.diag([5, 35, 10, 0, 0, 1, 1, 0.01]) # Weigths when in Energy mode
    #ocp.cost.W = np.diag([5, 35, 10, 0, 0, 1, 1, 0]) # Weights when not in Energy mode

    # Set QP solver and integration
    ocp.solver_options.tf = Tf
    ocp.solver_options.qp_solver_cond_N = N
    ocp.solver_options.qp_solver =  "FULL_CONDENSING_HPIPM"
    #"PARTIAL_CONDENSING_HPIPM" #"FULL_CONDENSING_QPOASES"
    ocp.solver_options.nlp_solver_type = "SQP"
    ocp.solver_options.hessian_approx = "GAUSS_NEWTON"
    ocp.solver_options.integrator_type = "ERK"
    # ocp.solver_options.sim_method_num_stages = 4
    # ocp.solver_options.sim_method_num_steps = 3

    # Create json file
    ocp_solver = AcadosOcpSolver(ocp, 'acados_ocp_' + ocp.model.name + '.json')

    return ocp_solver
```

Code listing C.4: main

```
def main():
    ocp_solver = energyMPC.ocpSolver()
    ocp_integrator = AcadosSimSolver(ocp_solver.acados_ocp, 'acados_ocp_' +
    ocp_solver.acados_ocp.model.name + '.json')
    Nsim = 100 # Number of iterations
    nx = ocp_solver.acados_ocp.model.x.size()[0]
    nu = ocp_solver.acados_ocp.model.u.size()[0]
    ny = nx + nu
    N = ocp_solver.acados_ocp.solver_options.qp_solver_cond_N
    Tf = ocp_solver.acados_ocp.solver_options.tf #Prediction horizon

    simX = np.ndarray((Nsim, nx))
    simU = np.ndarray((Nsim, nu))

    x0 = np.array([0, 1, 0, 0, 0, 0]) # Initial state
    x_cur = x0
    simX[0,:] = x0

    computation = []
    time_solve = 0
    for i in range(Nsim):

        ocp_solver.set(0, "lbx", x_cur)
        ocp_solver.set(0, "ubx", x_cur)

        start = time.time()
        solver_status = ocp_solver.solve() # Solve the ocp at current iteration
        t = time.time() - start
        computation.append(t)
        print(f'time:␣{t*1000:.2f}[ms],␣iter:␣{i}') #Calculate time it takes to solve iteration
        time_solve += t

        if solver_status != 0:
            print(f'solver␣error:␣{solver_status}')
            # raise Exception(f'solver error: {solver_status}')

        simU[i, :] = ocp_solver.get(0, "u")

        ocp_integrator.set("x", x_cur)
        ocp_integrator.set("u", simU[i, :])

        integrator_status = ocp_integrator.solve()

        if integrator_status != 0:
            raise Exception(f'integrator␣error:␣{integrator_status}')

        x_cur = ocp_integrator.get("x")
        simX[i, :] = x_cur


    print(f'avg.␣time:␣{time_solve/Nsim*1000}[ms]')
    print("Average␣speed:{}m/s".format(np.average(simX[:, 3])))

    # Plots

    t = np.linspace(0.0, Nsim * Tf / N, Nsim + 1)
    t_2 = np.linspace(0.0, Nsim * Tf / N, Nsim)

    plt.figure("Figure␣1")
    plotFnc.states(t_2, simX, simU)

    plt.figure("Figure␣2")
    plotFnc.computationalLoad(Nsim, computation)

    plt.figure("Figure␣3")
    plotFnc.energyPlot(t, simX)
    plt.show()


main()
```