CHALMERS
UNIVERSITY OF TECHNOLOGY

High speed lane following using predictive control

Development, implementation and evaluation of high speed lane following algorithms

Master's thesis in Systems, Control and Mechatronics

MARKUS CARLANDER
DANIEL PIHLQUIST

Department of Signals and Systems
CHALMERS UNIVERSITY OF TECHNOLOGY
Gothenburg, Sweden 2016

Master's thesis 2016:EX057

High speed lane following using predictive control

Development, implementation and evaluation of
high speed lane following algorithms

MARKUS CARLANDER
DANIEL PIHLQUIST

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
Gothenburg, Sweden 2016

High speed lane following using predictive control
Development, implementation and evaluation of
high speed lane following algorithms
MARKUS CARLANDER DANIEL PIHLQUIST

Supervisor: Martin Larsson, Delphi Automotive
Examiner: Jonas Fredriksson, Signals and Systems

Master's Thesis 2016:EX057
Department of Signals and Systems
Division of Systems and Control
Chalmers University of Technology
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Cover: Picture of the Volvo V40 test vehicle used for real world testing.

Gothenburg, Sweden 2016

High speed lane following using predictive control
Development, implementation and evaluation of
high speed lane following algorithms
MARKUS CARLANDER DANIEL PIHLQUIST
Department of Signals and Systems
Chalmers University of Technology

Abstract

The automotive industry is constantly trying to make cars safer. While automotive engineers try to add additional safety features to cope with human error, computer scientists try to remove the driver completely. Computers can do things a lot faster and more consistently than humans, provided that they have correct instructions and data. Moving towards a computer controlled car is a logical step. Therefore, the industry is striving towards building new functionality which replaces human involvement in driving.

In this thesis, the aim is to develop a lateral control algorithm for high speed lane following. One type of controller that has shown promising results in vehicle path following is the Model Predictive Controller. With knowledge about the system dynamics, the MPC can predict the system state in the future and optimize the control signal to a desired behavior. An MPC controller and a finite horizon linear quadratic regulator (LQR) controller are developed for comparison in performance. The controllers uses a similar approach to solve a convex optimization problem, but the MPC offers the ability to set actuator constraints. Both the MPC and the finite horizon LQR controller show promising results in simulation and real world tests.

Keywords: lateral control, vehicle dynamics, model predictive control, modeling and simulation.

Acknowledgements

Mattias Styren
Mats Björnerbäck
Joakim Rukin
Adam Pettersson
Viktor Svensson
Anton Jacobsson
Martin Larsson
Johan Fur-bar-yi

Markus Carlander
Daniel Pihlquist, Gothenburg, June 2016

Contents

Notation

Acronyms

LKA	Lane Keeping Aid
TJA	Traffic Jam Assist
ACC	Adaptive Cruise Control
LF	Lane Following
CT	Continuous Time
DT	Discrete Time

Capital letters

A	CT motion matrix
A_d	DT motion matrix
A_m	CT Mathematica generated motion matrix
B	CT input vector
B_d	DT input vector
B_m	CT Mathematica generated input vector
C_f	Front cornering stiffness
C_m	CT Mathematica generated output matrix
C_r	Rear cornering stiffness
F_{fxw}	Longitudinal force on front axle in front wheel coordinate system
F_{fyw}	Lateral force on front axle in front wheel coordinate system
F_{fz}	Vertical force on front axle
F_{rx}	Longitudinal force on rear axle
F_{ry}	Lateral force on rear axle
F_{rz}	Vertial force on rear axle
I_{zz}	Yaw inertia around the center of gravity
J	Cost function
L	Length between front and rear axle
M	Number of control updates (Control horizon)
N	Number of predictions (Prediction horizon)
Q	Weight matrix
R	Radius of the curvature
T_{MPC}	Sample time for the MPC

Lower case letters

a	Lane marker coefficients
a_x	Longitudinal acceleration
a_y	Lateral acceleration
c_0	Tire stiffness parameter
c_1	Tire stiffness parameter
g	Gravity
h	Discretization time
l_f	Length between front axle and center of gravity
l_r	Length between rear axle and center of gravity
m	Total mass of the vehicle
p_y	Lateral position at the center of gravity
r_i	References for trajectory
u_{max}	Maximum value on input signal
u_{min}	Minimum value on input signal
$\dot{u}_{max}$	Maximum rate on input signal
$\dot{u}_{min}$	Minimum rate on input signal
v_f	Velocity at the front axle
v_{fxw}	Longitudinal velocity at the front axle in front wheel coordinate system
v_{fyw}	Lateral velocity at the front axle in front wheel coordinate system
v_r	Velocity at the rear axle
v_{rx}	Longitudinal velocity at the rear axle
v_{ry}	Lateral velocity at the rear axle
v_x	Longitudinal velocity at the center of gravity
$\dot{v}_x$	Time derivative of longitudinal velocity
v_y	Lateral velocity at the center of gravity
$\dot{v}_y$	Time derivative of lateral velocity
x	State vector
z	Optimization vector

Greek letters

α	Lateral side slip angle at the center if gravity
α_{fy}	Lateral side slip angle at the front axle
α_{ry}	Lateral side slip angle at the rear axle
δ_f	Steering angle on front wheel
δ_{EPAS}	Input to the power steering interface
δ_{stw}	Steering wheel angle
δ_{ratio}	Linear relationship between δ_f and δ_{stw}
κ	Curvature of the road
ω_i	State weightings of the controller behavior
ω_z	Yaw rate at the center of gravity
ψ_z	Yaw angle at the center of gravity

1

Introduction

Since the dawn of the development of vehicles, car manufacturers have always tried to simplify the task of driving the car as well as making it safer. The first fully automatic gearbox was introduced by General Motors in 1940 [13]. Since then, a lot of other safety and comfort functions have been developed and added to cars, such as seat belts, anti-lock braking system, electronic stability control, adaptive cruise control and many more.

The post point of no return safety, i.e. a safety function that acts only when an accident is going to happen or already happening, is reaching its limit. There are only so many mechanical safety features one can add to the chassis structure before other factors such as aesthetic look, fuel consumption or road handling become compromised. Now that the cars of today are reaching this limit, car manufacturers are looking to completely exclude another factor from the equation: the factor of human error. When given some thought, it makes more sense to reduce the probability of getting into an accident, than to make the car more resilient to accidents.

While humans have sufficient capabilities to take in all the information necessary to read a traffic situation, we can still lose focus due to distractions, and subsequently fail to see upcoming dangers. We can all make errors and mistakes. Due to the human factor, car manufacturers are implementing so called *Advanced Driver Assistance Systems* or ADAS for short. ADAS intends to aid the driver in recognizing danger before it is too late and thereby warning or intervening, while at the same time taking the car dynamics into account. Computer systems can do this precise and exact, whereas humans can make estimates based on prior experience and our limited senses.

One of the key components in ADAS is the sensors which deliver information about the surrounding environment. Two sensor types that have an increase in use for the past few years are the camera and the radar. With their low cost and yet high capabilities, cameras and radars are used in many active safety systems today. They both have their strengths and weaknesses; the camera and associated algorithms can classify objects and detect visual features such as lane markings and traffic signs, whereas the exact range to and velocity of the object is difficult for the camera to estimate. That's where the radar comes in; with accurate range and range rate measurements, the relative position and velocity of other objects can be estimated. The radar can however not determine what kind of an object that reflected the radio waves. Using a mathematical method called sensor fusion, algorithms use the strengths of the two sensors to provide accurate measurements of the surrounding

environment.

ADAS functions can be separated into two categories; safety functions and comfort functions. Safety functions are there to minimize or prevent injuries when danger is imminent, whereas comfort functions take away some of the tedious tasks that comes with driving, such as following the lane on freeways, or constantly adjusting the speed when catching up to other vehicles. In 1992 Mitsubishi introduced a crude predecessor to the forward collision warning (FCW) system, which used a lidar based system to alert the driver when the distance to the leading car was too small [12]. In today's automotive industry, some of these functions are already implemented in production cars, such as Adaptive Cruise Control (ACC), Traffic Jam Assist (TJA) and Lane Following (LF). While none of these functions are capable of completely controlling the car, they will still play an imperative role in the development of autonomous cars.

In today's automotive industry, autonomous driving technology is becoming an increasingly popular investment for car manufacturers, and the race towards the fully autonomous car is ongoing. In 2004, DARPA (Defense Advanced Research Projects Agency) launched its autonomous driving challenge called *DARPA Grand Challenge*, in which universities and companies got the opportunity to develop their own autonomous vehicle, and have it compete in a number of different challenges [3]. Since then, many companies have joined the race for the first autonomous car, with giants such as Google, Baidu and Volvo amongst them [8][4].

There are many different functions required for autonomous driving, with different logic handling different scenarios and situations. We as humans adapt to the situation that we are in; if we are driving in an urban area, we know that we must stay alert and focus on many different elements such as other cars, obstacles and pedestrians. If we are driving on the highway however, the scenario is pretty static, and we only need to look ahead for braking cars and follow the curvature of the road. For an autonomous car these different scenarios will demand different logic and different controllers.

Lateral and longitudinal control are the two key concepts of autonomous cars. While there are several algorithms that measures the environment and makes decisions based on this information, in the end it all comes down to the control of the vehicle motion. There are many different approaches to the lateral control of a highway vehicle. Some suggest a sliding mode control approach [10] to achieve avoid oscillations around the lane center. There has also been research in H_∞ control [17] which allows for model uncertainties. This report presents a comparison between two different methods to control the vehicle laterally on freeways; namely the Model Predictive Controller and the Finite Horizon Linear Quadratic Controller.

1.1 Problem formulation

This project aims to develop, implement and finally evaluate several different control methods that controls a vehicle laterally at speeds above 70 km/h such that it follows the lane on highways.

The controller will use a set of inputs and one output. The inputs to the controller will consist of the road information and the vehicle state information. The road information can be divided into two parts. One part describes the lateral offset y from the road tangent line, as well as the angular offset θ from the road tangent line. The other part is how the road is changing, which is described by the current radius of curvature R. The controller will also use vehicle states such as lateral and longitudinal velocity v_y and v_x as well as yaw rate ω_z are used as input to the system. Given all of these inputs, the controller shall calculate the steering wheel angle which makes the car follow the road.

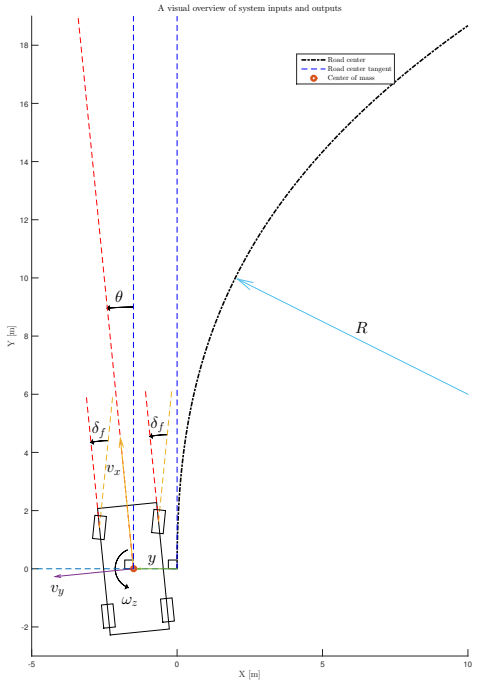


Figure 1.1: Over view of the vehicle reference and angles.

Additionally, a set of issues are also to be answered.

- What kind of controller is best suited for this system?
- What kind of demands does this controller put on the sensor system?
- What kind of demands does this controller put on the actuator system?
- How realistic is it to implement the different controllers, in terms of computational complexity, on existing embedded hardware in automotive systems?

1.2 Limitations and Assumptions

A number of things are excluded in this project due to limitations in time and physical availability.

- **Vertical dynamics**
  The vertical dynamics is the movement of the car along the axis perpendicular to the road plane. The vertical curvature of Swedish highways is relatively low, and the vertical dynamics is therefore negligible. It is also difficult to model, as it is only measured instantaneously, i.e. where the car is. If it could be estimated at some distance ahead, such that some sort of feed forward in the control system could be implemented, it would be easier to compensate for it.

- **Roll/pitch dynamics**
  Neither the roll or pitch angle nor their respective angular velocities can be measured accurately. The fast varying pitch and roll dynamics are due to local road imperfections. The roads are assumed to be made even enough to neglect roll/pitch dynamics.

- **Longitudinal dynamics**
  The change in longitudinal velocity on freeways is usually small, such that it can be considered to have no or little impact on lateral and rotational dynamics. The control of the longitudinal velocity will be performed by the built in cruise controller.

- **Tire nonlinearities**
  Tire dynamics is difficult to model and including them leads to a more complex controller which is computationally heavier. The controller is however only designed to work on freeways, where the lateral velocity and acceleration is low, and little or no longitudinal dynamics. Therefore, the tires will operate in the linear region.

- **All sensor data is filtered and will not be processed any further**
  As the current road sensing system is under development, and is already sent through a number of filters, further filtering might decrease its quality. Thus, it was decided that the sensor data should not be filtered any further.

- **Lateral gradient of the road**
  All roads are made with a lateral gradient in order to decrease the lateral acceleration when taking a curve. A lateral gradient leads to better curve taking. Since the roll angle is not measured, and the lateral gradient is not included in the mathematical model, the lateral gradient will be neglected.

- **Low curvature**
  The curvature of the road induces errors in the modeling when the model is sampled and discretized. However, it will be shown that the curvature is low enough such that this error can be neglected.

- **Linear coupling between steering wheel angle and tire angles**
  The ratio between the steering wheel angle and the tire angles is not linear over the entire range of the steering wheel. For small angle it is however fair to assume that they are. Another reason why this is neglected is that there is no information available about the relationship between them for the used test vehicle.

- **Steering wheel dynamics**
  The steering wheel actuator is assumed to behave as a first order linear time invariant system.

1.3 A few words about road geometry

In the previous section there are a number of limitations and assumptions that are listed. There is one of them which defines and binds the problem, namely the one which states that assumes a low curvature for the roads.

Below is a table showing maximum allowed curvature for Swedish roads. Driving in a certain curvature with the intended velocity, this results in a lateral acceleration according to $a_y = \kappa v_x^2$, where κ is the curvature, v_x is the velocity along the road tangent and a_y is the lateral acceleration, which is perpendicular to the road tangent.

Velocity [km/h]	Radius of curvature [m]	Max lateral acceleration [m/s^2]
120	1200	0.93
110	900	1.04
100	700	1.10
80	400	1.23
60	250	1.98

2

Modeling

This chapters derive and define a mathematical model that describes the system dynamics. The model is then linearized around the operating points, and then discretized for an arbitrary sampling time.

2.1 Mathematical Model Formulation

A vehicle dynamics model have to be selected so that the controllers can have knowledge about the system and make correct estimations of future states. The mathematical model is desired to be detailed enough to describe the dynamics of the real vehicle. On the other hand will a high detailed model involve more state variables, more model parameters and also more complex calculations for the controller to preform. There is a trade of between the accuracy of a detailed model and the small computation effort for a less detailed model.

The lateral dynamics of a vehicle is highly non-linear and is affected of both the longitudinal and the vertical motion. The non-linearty of the lateral dynamics is much affected of the dynamics of the tires, which is connected to the slip between tire and ground. The lateral dynamics is also involving trigonometrical functions which also contributes to the non-linear behavior, [1].

A bicycle model is a quite simple model since it neglects lateral load distribution and hence neglects the roll dynamics in cornering. A bicycle model does only need two states variables, lateral velocity (v_y) and yaw rate (ω_z) in order to describe the lateral motion as described in [9]. The number of needed state variables are affecting the computational effort of calculating the optimal steering angle and the simplicity of the bicycle model is highly valued. The accuracy of the bicycle model is considered to be good enough to let a controller use it in order to estimate the future motion.

The bicycle model is still highly non-linear even though it simplifies the lateral dynamics. To handle this, further simplifications and assumptions can be made when constructing the mathematical model as stated below:

- Since the lane centering algorithm is to be developed for high speed motorways then the radius of the curvature will be much larger than the length of the vehicle. Therefore can forces be approximated to be either lateral or longitudinal directed in the vehicle on-board coordinate system.

- High longitudinal velocity makes it possible to assume that the yaw rotation of the vehicle will be around the center of gravity.
- If the longitudinal velocity is assumed to be held constant for the entire prediction horizon then the longitudinal acceleration will be small and can be neglected. This induce that the bicycle model can be expressed with only two states, v_y and ω_z as described above.

2.1.1 Equilibrium equations

The mathematical model for the bicycle model can be developed by constructing the equilibrium equations from the figure 2.1. The figure shows all forces, angles, velocities and lengths that are needed to generate the complete mathematical model. The black and white circle shows the center of gravity (COG), the front is located to the right in the figure. The figure have two different coordinate systems, one is expressing the vehicle frame and is located in the COG, the other one is posed in the front wheel center and is sub-scripted with a w.

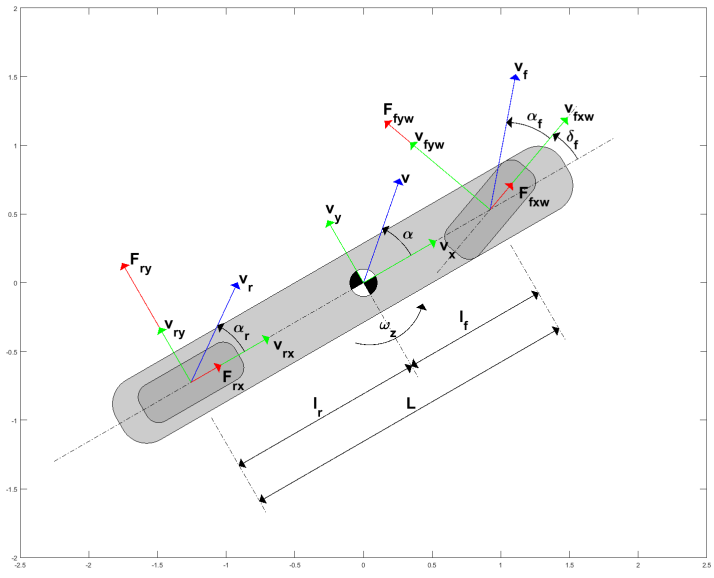


Figure 2.1: The figure shows the forces (red arrows), velocities (green and blue arrows), angles and angular velocities (curved arrows) working on the system. Note that angles are drawn large for clear visualization but are assumed to be small.

According to the notation in figure 2.1, equilibrium equations for the bicycle model can be derived as:

$$ma_x = F_{fxw}\cos\delta_f - F_{fyw}\sin\delta_f + F_{rx}, \tag{2.1}$$

$$ma_y = F_{fxw}\sin\delta_f + F_{fyw}\cos\delta_f + F_{ry}, \tag{2.2}$$

$$I_{zz}\dot{\omega}_z = (F_{fxw}\sin\delta_f + F_{fyw}\sin\delta_f)\, l_f - F_{ry}l_r, \tag{2.3}$$

where the mass of the vehicle is denoted m and the yaw inertia is noted as I_{zz}.

The accelerations in either direction (x and y), can be expressed as a relation between velocity derivative and the yaw rate of the vehicle as:

$$a_x = \dot{v}_x - \omega_z v_y, \tag{2.4}$$

$$a_y = \dot{v}_y + \omega_z v_x. \tag{2.5}$$

2.1.2 Vertical forces

Cornering induces the vehicle to roll which imply that the load distribution changes between right and left side of the vehicle. In the same manner as longitudinal acceleration and deceleration induces the vehicle to pitch and change the load distribution between front and rear axle.

Since the bicycle model is neglecting the roll dynamics and that the longitudinal velocity is assumed to be constant in the prediction horizon, the vertical forces will be constant and the load distribution between front and rear axles will be directly proportional to the distance between the axles and the COG as:

$$F_{fz} = \frac{mgl_r}{L}, \tag{2.6}$$

$$F_{rz} = \frac{mgl_f}{L}. \tag{2.7}$$

2.1.3 Lateral forces

The lateral forces, F_{ry} and F_{ry}, on the vehicle can be formulated as a combination between the lateral side slip angle, (α_r and α_f), of the vehicle and a parameter denoted as a tire cornering stiffness (C_r and C_f) as:

$$F_{ry} = -C_r\alpha_r, \tag{2.8}$$

$$F_{fyw} = -C_f\alpha_f. \tag{2.9}$$

Where the tire cornering stiffness is the derivative of the lateral force with respect to the slip angle and describes the dynamics of a tire for specific slip angle, [9]. In other words is the tire cornering stiffness varying for different slip angles and for different tires.

The slip angle can be expresses as a ratio between the lateral and longitudinal velocity of the wheels as:

$$\alpha_f = \arctan\left(\frac{v_{fyw}}{v_{fxw}}\right) = \arctan\left(\frac{(v_y + l_f\omega_z)\cos\delta_f - v_x\sin\delta_f}{(v_y + l_f\omega_z)\sin\delta_f + v_x\cos\delta_f}\right), \tag{2.10}$$

$$\alpha_r = \arctan\left(\frac{v_y - l_r\omega_z}{v_x}\right). \tag{2.11}$$

The true tire cornering stiffness have a non-linear characteristic which is affected by many parameters like for example vertical load, steering angle, longitudinal velocity, tire friction. For small slip angles is the non-linearity small enough to be negligible but increases when side slip grows, [11]. There exist many different tire modeling approaches, some are based on pure mathematics where as other approaches are more or less empirical. The mathematical modeling is restricted to steady state condition where as the empirical modeling uses exponential, parabolic or arctangent functions with more or less success to adapt the tire characteristic, [15]. Since the slip angles are assumed to be small, the front respectively rear cornering stiffness be expressed with a less advanced empirical model. The front cornering stiffness and rear cornering stiffness can be expressed as:

$$C_f = c_0 F_{fz} + c_1 F_{fz}^2, \tag{2.12}$$

$$C_r = c_0 F_{rz} + c_1 F_{rz}^2, \tag{2.13}$$

with use of the model presented in [5].

Since the turning angle and the slip angles both are assumed to be small can the trigonometrical functions be approximated as $\cos\beta \approx 1$, $\sin\beta \approx \beta$ and $\arctan a \approx a$. The equilibrium equations (2.1) to (2.3) and the slip angle equations, (2.10) and (2.10), can be rewritten as follows:

$$ma_x \approx F_{fxw} - F_{fyw}\delta_f + F_{rx}, \tag{2.14}$$

$$ma_y \approx F_{fxw}\delta_f + F_{fyw} + F_{ry}, \tag{2.15}$$

$$I_{zz}\dot{\omega}_z \approx (F_{fxw}\delta_f + F_{fyw}\delta_f)\, l_f - F_{ry} l_r, \tag{2.16}$$

$$\alpha_f \approx \frac{v_y + l_f\omega_z - v_x\delta_f}{(v_y + l_f\omega_z)\delta_f + v_x}, \tag{2.17}$$

$$\alpha_r \approx \frac{v_y - l_r\omega_z}{v_x}. \tag{2.18}$$

Since the longitudinal velocity is expected to be constant during the entire prediction horizon, the longitudinal acceleration can be assumed to zero, $\dot{v}_x \approx 0$. Further, the longitudinal tire forces are therefore assumed to be small and can be neglected from the equations of motion. The longitudinal motion (2.1) can be rewritten as:

$$F_{fwx} + F_{rx} = -m\omega_z v_y - C_f \frac{v_y + l_f\omega_z - v_x\delta_f}{(v_y + l_f\omega_z)\delta_f + v_x}\delta_f \approx 0. \tag{2.19}$$

By combining equations (2.8) to (2.9) and (2.15) to (2.18). The lateral dynamics can finally be expressed as:

$$\dot{v}_y = -\frac{C_f}{m}\frac{v_y + l_f\omega_z - v_x\delta_f}{(v_y + l_f\omega_z)\delta_f + v_x} - \frac{C_r}{m}\frac{v_y - l_r\omega_z}{v_x} - \omega_z v_x, \tag{2.20}$$

$$\dot{\omega}_z = -\frac{C_f}{I_{zz}}\frac{v_y + l_f\omega_z - v_x\delta_f}{(v_y + l_f\omega_z)\delta_f + v_x}\delta_f l_f + \frac{C_r}{I_{zz}}\frac{v_y - l_r\omega_z}{v_x}l_r. \tag{2.21}$$

2.1.4 State variable extension

In addition to lateral velocity and yaw rate, the vehicle need to have state variables that can be compared with a reference trajectory. The reference trajectory, which is described in chapter 3.1 is needed to convert from the current lane marker measurement to a set of reference states that the vehicle is desired to achieve in the future. The key is to express states that can be computed from the mathematical model's state variables and the lane marker estimation so that a set of predicted states can be compared to set of reference states.

For this, lateral position p_y and yaw angle ψ_z are a good choice since it can be easily computed from the mathematical model state variables as well as the lane marker estimation.

The yaw angle ψ_z can be derived by integrating the yaw rate ω_z as:

$$\dot{\psi}_z = \omega_z. \tag{2.22}$$

The lateral offset on the other hand cannot be derived by simply integrating the lateral velocity v_y. Because the lateral offset of all predictions should always expressed from the vehicle's current state on-board coordinate system and not from the predicted on-board coordinate system. The lateral offset can be expressed as:

$$\dot{p}_y = v_y \cos\psi_z - v_x \sin\psi_z. \tag{2.23}$$

2.2 Linearization

The motion equations are non-linear and needs to be linearized in order to use a linear control frame work. The model can be linearized around a operating point by using first order Taylor expansion. The linearized model is only valid for small deviation from the operation point.

If the operation values are defined as x_0 for the state variables and u_0 for the input signal, (δ_f), then the non-linear function can be expressed as:

$$\dot{x}_0 = f(x_0, u_0). \tag{2.24}$$

The resulting Taylor series expansion can therefore be expressed as:

$$\dot{x}(t) \approx f(x_0, u_0) + \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{x=x_0\\u=u_0}} (x(t) - x_0) + \left.\frac{\partial f(x,u)}{\partial u}\right|_{\substack{x=x_0\\u=u_0}} (u(t) - u_0)\,. \quad (2.25)$$

Where the current state vector is denoted $x(t)$ and the current input signal $u(t)$ acting on the mathematical model.

The equilibrium point for the lateral dynamics is when the vehicle is traveling forward with no lateral motion. If the lateral dynamics is linearized around equilibrium then the operation points $x_0 = (0 \dots 0)$ and $u_0 = 0$. Then the model matrix A and input vector B can be formulated as:

$$A = \left.\frac{\partial f(x,u)}{\partial x}\right|_{\substack{x=x_0\\u=u_0}}, \quad (2.26)$$

$$B = \left.\frac{\partial f(x,u)}{\partial u}\right|_{\substack{x=x_0\\u=u_0}}. \quad (2.27)$$

Since the linearization is done in a steady state which imply that the actual derivative:

$$\dot{x}_0 = f(x_0, u_0) = 0. \quad (2.28)$$

The linear model can therefore be described by the following state space representation:

$$\dot{x}(t) = Ax(t) + Bu(t). \quad (2.29)$$

The model matrix and the input vector can be seen in Appendix C.1 where the input $u(t)$ is the steering angle on the front wheel.

2.3 Power steering interface extension

The input to the current mathematical model is the steering angle of the front wheel but the controller is connected to the input of the power steering interface. The power steering interface *Electric Power Assisted Steering*, (EPAS) which is described latter in chapter 4.2 have dynamics between the requested steering angle and the actual steering angle. This means that the mathematical model have to be extended with the dynamics of the power steering interface.

Given the current mathematical model in Laplace domain, the extended mathematical model can be formulated as:

$$sx(s) = Ax(s) + B\delta_f(s), \quad (2.30)$$

$$= Ax(s) + BG_{EPAS}(s)\delta_{EPAS}(s). \quad (2.31)$$

where $G_{EPAS}(s)$ denotes the dynamics of the power steering system and $\delta_{EPAS}(s)$ represent the requested steering angle which the controller can be connected to.

The new mathematical model is desired to be expressed on the standard state space form:

$$\dot{x}(t) = Ax(t) + B\delta_{EPAS}(t), \tag{2.32}$$

and (2.31) must hence be reformulated so that G_{EPAS} is integrated in the new model matrices.

The power steering interface of the real test vehicle can be estimated to a first order filter with time constant 0.4 s as can be seen in chapter 6.1.1. Therefore the EPAS dynamics can be formulated as:

$$\dot{\delta}_f(t) = -\frac{1}{0.4}\delta_f(t) + \frac{1}{0.4}\delta_{EPAS}(t). \tag{2.33}$$

By extending the state vector as

$$x_e(t) = \begin{pmatrix} x(t) \\ \delta_f(t) \end{pmatrix}, \tag{2.34}$$

an approach to achieve the desired state space representation is to augment the model as:

$$\begin{aligned} \dot{x}_e(t) &= \begin{pmatrix} A & B \\ 0 & -\frac{1}{0.4} \end{pmatrix} x_e(t) + \begin{pmatrix} 0 \\ \frac{1}{0.4} \end{pmatrix} \delta_{EPAS}(t), \\ x(t) &= \begin{pmatrix} I & 0 \end{pmatrix} x_e(t). \end{aligned} \tag{2.35}$$

This approach requires the mathematical model to be extended with an extra state variable and therefore was another approach investigated. By first expressing (2.29) as a transfer function in Laplace domain and then multiply with $G_{EPAS}(s)$ gives a transfer function of the complete dynamics. Which can be converted to state space representation by use of the StateSpaceModel function in Mathematica, [16]. This gives a automatic generated state space model which can be expressed as:

$$\begin{aligned} \dot{x}_m(t) &= A_m x_m(t) + B_m \delta_{EPAS}(t), \\ x(t) &= C_m x_m(t). \end{aligned} \tag{2.36}$$

where x_m denotes a auto generated state vector. The C_m matrix makes it possible to convert between the automatic generated state space vector x_m and the previous selected state vector x. This approach also resulted in a state vector with five states but will be used in the MPC controller whereas the FHLQC controller will make use of the first approach. The model matrices can be seen in Appendix C.2.

2.4 Discretization

The two methods of control that will be described later in assumes that the model is expressed in discrete time. Usually, the sampling time is constant between different samples.

The lateral reference for the road ahead is a function of the longitudinal distance. It is desired to sample the lateral reference at equidistant points along this reference function. As such, the predicted path will be sampled in space, and not in time. However, since the longitudinal velocity is assumed to be constant, the relation between the sampling time and sampling distance is linear and can be expressed as:

$$h = \frac{ds}{v_x}, \tag{2.37}$$

where h is the sampling time, ds is the sampling distance and v_x is the longitudinal velocity.

2.4.1 Forward Euler

Forward Euler is a simple method that implies that the future state is calculated with a predefined discretization time h from the current state. If the current state and input signal are assumed to be constant during the the discretization step. The forward Euler can be formulated as shown in the two following equations:

$$\dot{x}(k) \approx \frac{x(k+1) - x(k)}{h} = Ax(k) + Bu(k), \tag{2.38}$$

$$x(k+1) = \underbrace{(I + hA)}_{A_d} x(k) + \underbrace{hB}_{B_d} u(k), \tag{2.39}$$

where the discretized model matrix is denoted A_d and the discretized input vector is denoted B_d.

Forward Euler discretization have a drawback, the discretization time is affecting the stability of the discretized model. This can easily be seen in the following example when the control input is zero for all future states. If $x(k)$ is the current state then the next state is given by:

$$x(k+1) = (I + hA)\, x(k) \tag{2.40}$$

and the next state after that is given by:

$$\begin{aligned} x(k+2) &= (I + hA)\, x(k+1), \\ &= (I + hA)\, (I + hA)\, x(k), \\ &= (I + hA)^2\, x(k). \end{aligned} \tag{2.41}$$

This means that if $|I + hA| > 1$, future states will diverge to infinity when time increases and hence must the eigenvalues of the discrete time model be inside or on the border of the unit circle to guarantee stability, [7]. Therefore can the stability criterion be checked with the formulation:

$$|1 + h\lambda_{max}| \leq 1 \tag{2.42}$$

where λ_{max} represent the eigenvalue with greatest magnitude of the continuous time model.

2.4.2 Exact discretization

While the forward Euler is fast and easy to implement, it introduces some discretization error due to the finite difference approximation. To perform an exact discretization, one has to study the solution of a linear time invariant (LTI) system. The general LTI system has the form:

$$\dot{x}(t) = Ax(t) + Bu(t). \tag{2.43}$$

The continuous time solution of this is:

$$x(t) = e^{At}x(t_0) + \int_{t_0}^{t} e^{A(t-\tau)}Bu(\tau)d\tau, \tag{2.44}$$

where $x(t_0)$ is the initial condition. Since we wish to study the system in discrete time, i.e. in fixed time intervals, we look at $x(t)$ between $x(kT)$ and $x(kT + T)$, where T is the sampling time and $k \in \mathbb{Z}$. It is assumed that $x(kT)$ is known at time instance kT.

$$x(kT + T) = e^{AT}x(kT) + \int_{kT}^{kT+T} e^{A(kT+T-\tau)}Bu(\tau)d\tau. \tag{2.45}$$

If the control inputs are assumed to be constant between every sampling instance, the control signals can be moved outside of the integral, and the integral can be computed:

$$x(kT + T) = e^{AT}x(kT) + A^{-1}(e^{AT} - I)Bu(kT). \tag{2.46}$$

If we define that $x_d(k) = x(kT),\ k \in \mathbb{Z}$, we can therefore define our discrete matrices as:

$$\begin{aligned} A_d &= e^{AT}, \\ B_d &= A^{-1}(e^{AT} - I)B, \end{aligned} \tag{2.47}$$

and finally write it on standard discrete state space form:

$$x_d(k+1) = A_d x_d(k) + B_d u_d(k). \tag{2.48}$$

2.4.3 The consequences of curvature

As mentioned in the beginning of section 2.4, the road is sampled in space at points equidistantly spaced from each other, along an axis parallel to the vehicle heading. The sampling is done in a global coordinate system, located at the COG during the initial time instance, and not in the curved coordinate system which follows the road reference. Due to the fact that the car changes heading, the distance change ΔY along the global Y axis is not constant, and is therefore different over all time instances. Therefore, a sampling error is induced. This is illustrated in figure 2.2.

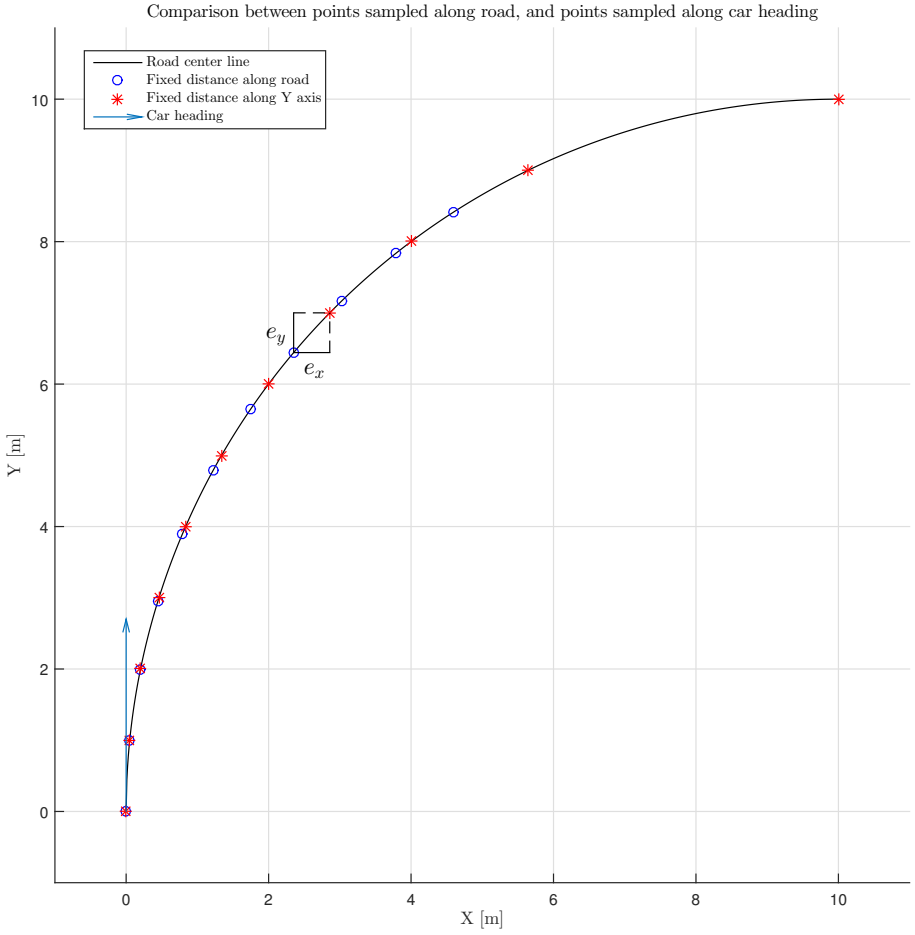


Figure 2.2: An excessivly high curvature to illustrate the induced errors.

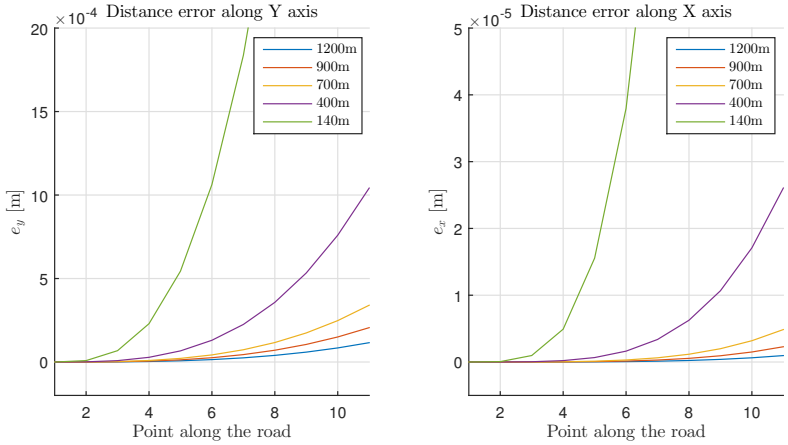


Figure 2.3: The errors in X and Y plotted over increasing distance away from the vehicle.

In Figure 2.3 the distance error in both the X and Y direction is observed for sampled points along the road. The error increases exponentially with increased distance from the car. The figure does however show that with increasing radius of curvature of the road, these errors decrease. This justifies the fact that the road can be sampled in this way, without introducing any substantial errors that propagate through the rest of the modeling.

3

Control design

The control design follows the modeling of the system. This chapter will briefly describe how the reference trajectory is defined. Later on, a general formulation of Model Predictive Control will be presented, and then go into the actual method and implementation. After this the Finite Horizon Linear Quadratic Controller will be presented, which has the same basic principle but the finds control signal in a different way.

3.1 Reference Trajectory

The reference trajectory is expressed in a two dimensional space domain with the coordinate system placed in the COG. Positive values on x-axle is placed in front of the COG where as positive values on y-axle is placed on the right side of COG. The reference points can be seen as a local coordinate systems that needs to be expressed in the vehicle on-board coordinate system. To do so, the longitudinal distance, lateral distance and yaw angle is parameters that have to be calculated from the lane marker estimation. The number of reference points is the same as the number of predictions for the controller and the idea is that each prediction should match the reference points.

3.1.1 Longitudinal sampling of the road

Since the lateral position of the center line is a function in terms of the longitudinal position, the first step in the reference mapping is to establish the longitudinal positions of the reference points. The longitudinal distance between each reference point is set to always be evenly distributed but not necessary the same for all longitudinal velocities. The longitudinal spacing between reference points can be done with two different strategies.

Fixed distance between the each sample: Since the longitudinal distance is fixed, the discretization time step have to vary according to different velocities. Small longitudinal velocities results in a high discretization time steps and the other way around. The drawback with this strategy is that the discretization stability criterion (2.42) might limit the distance to be unnecessary short for high velocities.

Fixed discretization time between each sample: In this way the distance between each sample will be the varying factor. A small longitudinal velocity results in a short distance between each sample. This strategy is beneficial since the prediction horizon distance have to be longer for higher longitudinal velocities.

Since the latter (fixed discretization time) is more beneficial, this strategy will be used in the controllers and the longitudinal reference vector can be expressed as:

$$d(k+1:N) = \begin{pmatrix} v_x h & v_x 2h & \dots & v_x N h \end{pmatrix}. \tag{3.1}$$

3.1.2 Lateral and yaw reference

The estimation of the lane center is given as a third degree polynomial:

$$\begin{aligned} r_{p_y}(k+i) = a_0 + a_1 d(k+i) + a_2 d(k+i)^2 + a_3 d(k+i)^3, \\ i \;=\; 1,\; 2,\; \dots,\; N, \end{aligned} \tag{3.2}$$

and represents the lateral offset reference of the longitudinal sampling of the road.

The yaw angle can be calculated from the tangents of the center line polynomial at the reference points. Since the derivative of a function is the slope at every point, i.e. $\frac{dy}{dx}$, taking the arctan of the derivative gives the angle at each point. By first calculating the derivative of the center line polynomial, the angle at each reference point is calculated by:

$$\begin{aligned} r_{\psi_z}(k+i) = \arctan\left(a_1 + 2a_2 d(k+i) + 3a_3 d(k+i)^2\right), \\ i \;=\; 1,\; 2,\; \dots,\; N. \end{aligned} \tag{3.3}$$

3.1.3 Curvature Feed Forward

The curvature (κ) of the reference trajectory can also be used for estimating the needed steering angle. For any arbitrary continuous function f, the curvature at some point d can be calculated by:

$$\kappa(d) = \frac{f''(d)}{\left(1 + (f'(d))^2\right)^{3/2}}, \tag{3.4}$$

where $f'(\cdot)$ and $f''(\cdot)$ are the first and second order derivative respectively. The reference function $r_{p_y}(\cdot)$ will in this case be a third degree polynomial, such that the curvature changes along the longitudinal axis. The curvature for the entire distance ahead is therefore approximated as an average of the curvature at N_κ points:

$$\kappa_a = \frac{1}{N_\kappa} \sum_{i=1}^{N_\kappa} \kappa(d_i), \tag{3.5}$$

where d_i are some points along the longitudinal axis placed equidistantly apart.

In [9], the author states that the required steering angle, $\delta_{f_{rec}}$, can be formulated as:

$$\delta_{f_{rec}} = \frac{C_f C_r L^2 + (C_r l_r - C_f l_f) m v_x^2}{C_f C_r L + (C_r l_f + C_r l_r) F_{fxw}} \frac{\omega_z}{v_x}, \tag{3.6}$$

and by assuming that the following approximations are valid:

$$a_y \approx \omega_z v_x, \tag{3.7}$$

$$\frac{\omega_z}{v_x} \approx \kappa_a, \tag{3.8}$$

$$F_{fxw} \approx 0. \tag{3.9}$$

The required steering angle approximation can therefore be expressed as:

$$\delta_{f_{req}} \approx L\kappa_a + \frac{C_r l_r - C_f l_f}{C_f C_r L} m v_x^2 \kappa_a, \tag{3.10}$$

according to the author of [9].

The steering angle feed forward is approximated to be the same for the entire control horizon. This can be written as:

$$r_u(k+0:M-1) = \begin{pmatrix} d_{f_{req}} & d_{f_{req}} & \dots & d_{f_{req}} \end{pmatrix}, \tag{3.11}$$

where r_u is the feed forward approximation of the required steering angle $\delta_{f_{req}}$.

3.2 Model Predictive Control

Model Predictive Control (MPC) is a control scheme developed in the 1970s in the chemical processing industry. By using a discrete and linear model to predict future states of the system, an optimization problem is solved where future states and control signals are optimization variables. The objective function is defined as the sum of the weighted square the state error for all prediction time instances. There are also linear constraints on the optimization variables which says that they must obey the state dynamics, and that the control signals and/or states must not exceed certain values. Due to the cost function being quadratic and the constraints being linear, MPC is a linearly constrained quadratic programming problem.

Due to the predictive nature of MPC, it can also take a changing future reference into account. Two parameters that defines the MPC problem is the prediction horizon N and the control horizon M. The prediction horizon decides how many samples into the future the problem shall be defined for. The control horizon decides how many of the future control moves that are used as optimization variables. In order to save computational load, the control horizon is often chosen to be shorter than the

prediction horizon. Thus the last control signal $u(k+M-1)$ lasts for the rest of the prediction horizon. The number of optimization variables is equal to $N \cdot n + M \cdot m$, where n is the system order and m is the number of control inputs.

The cost function for N future states and M future control signals can generally be defined as:

$$J = \sum_{j=1}^{N} \sum_{i=1}^{n} \omega_i \left(x_i(k+j) - r_i(k+j) \right)^2 + \sum_{l=0}^{M-1} \omega_u \left(u(k+l) - r_u(k+l) \right)^2, \tag{3.12}$$

where $r_i(k+j)$ is the reference for each state ($i = 1,\ 2, \ldots, n$) at each corresponding time instant ($j = 1,\ 2, \ldots, N$) and ω_i is the weight for each state variable in the left most summation. In the right most summation denotes $r_u(k+l)$ feed forward of a approximate input at the corresponding time instant ($l = 1,\ 2, \ldots, M-1$) and ω_u the weight on the input signal.

The discrete time state space:

$$x(k+1) = Ax(k) + Bu(k) \tag{3.13}$$

of the system represent a set of equality constrains between the future states, the current state and the control input. These equality constraints in combination with the previously stated cost function makes it possible to express the minimization problem as shown in (3.14). It also possible to add inequality constraints such as lower bounds, u_{lb}, respectively upper bounds, u_{ub}, on the input signal and deviation limits between time instances on the input signal Δu_{max} for example.

$$\min_{x,u}$$

$$J = \sum_{j=1}^{N} \sum_{i=1}^{n} \omega_i \left(x_i(k+j) - r_i(k+j) \right)^2 + \sum_{j=0}^{M-1} \omega_u \left(u(k+j) - r_u \right)^2$$

subject to

$$\begin{aligned}
x(k+1+i) &= Ax(k+i) + Bu(k+i),\ i = 0,\ 1, \ldots, M-1 \\
x(k+1+i) &= Ax(k+i) + Bu(k+M-1),\ i = M, M+1, \ldots, N-1 \\
u(k : k+M-1) &\leq u_{lb} \\
u(k : k+M-1) &\geq u_{ub} \\
|u(k+1 : k+M-1) - u(k : k+M-2)| &\leq \Delta u_{max}
\end{aligned} \tag{3.14}$$

Now that the optimization problem has been defined, it needs to be reformulated to a more suitable form. The algorithms that solves quadratic programming problems, usually referred to as *solvers*, need the cost function and constraints to be defined on matrix form, often refereed to as standard form, where matrices describe the problem. For the predicted states $x(k+1 : N)$ and the future control signals $u(k+0 : M-1)$, the optimization vector is defined as:

$$z = \begin{bmatrix} x(k+1) \\ x(k+2) \\ \vdots \\ x(k+N) \\ u(k) \\ u(k+1) \\ \vdots \\ u(k+M-1) \end{bmatrix}. \tag{3.15}$$

The optimization problem, (3.14), can now be described as:

$$\begin{aligned} \min_{x} \quad & \frac{1}{2} z^T H z + z^T f \\ \text{subject to} \quad & A_{eq} z = b_{eq} \\ & A_{in} z \geq b_{in} \end{aligned} \tag{3.16}$$

where H, f, A_{eq}, b_{eq}, A_{in} and b_{in} are constant matrices appropriately designed such that they uphold the cost function and the constraints. Using this quadratic formulation, the minimization problem can be solved using quadratic programming solver, which there are a number of fast and powerful implementations of.

3.2.1 MPC Implementation

The MPC controller can be visualised as shown in the figure 3.1. Given the inputs, the MPC block implements five main tasks in order to calculate the new steering angle. These tasks are:

- Calculate the reference trajectory; $r_i(k+j)$.
- Calculate approximate steering angle feed forward; $r_u(k+l)$.
- Calculate the mathematical model matrices; A_m, B_m and C_m.
- Construct the optimization matrices; H, f, A_{eq}, b_{eq}, A_{in} and b_{in}.
- Solve the minimization problem and return the new steering angle; $u(k)$.

The MPC make use of both references for the each state $r_i(k+j)$ where $i \in (p_y, \psi_z)$ and a approximate steering angle feed forward $r_u(k+l)$. Calculating these are the first task that the MPC controller performs, naturally because these have to be calculated in order to construct the H matrix and f vector, (3.14).

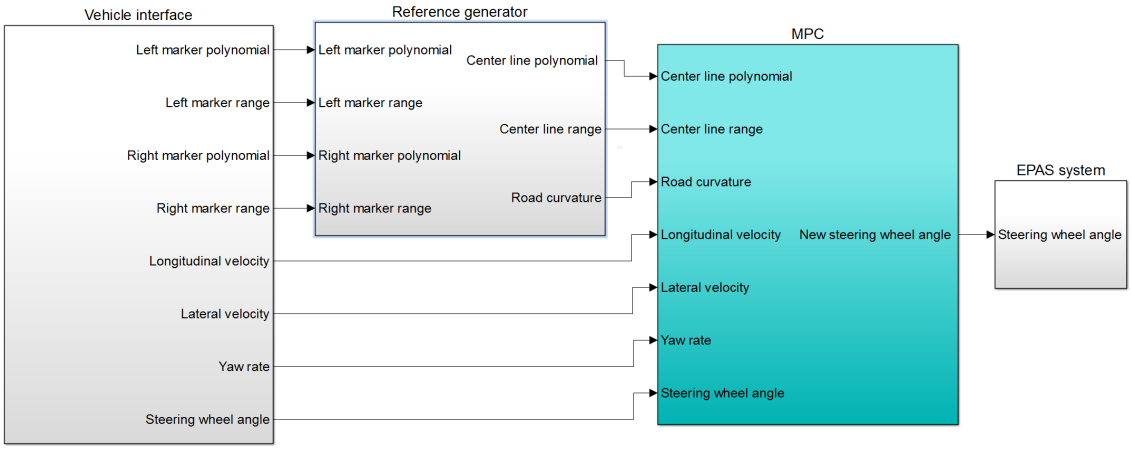


Figure 3.1: The cyan colored block represent the MPC implementation. The controller make use of seven input signals in order to compute a new steering angle.

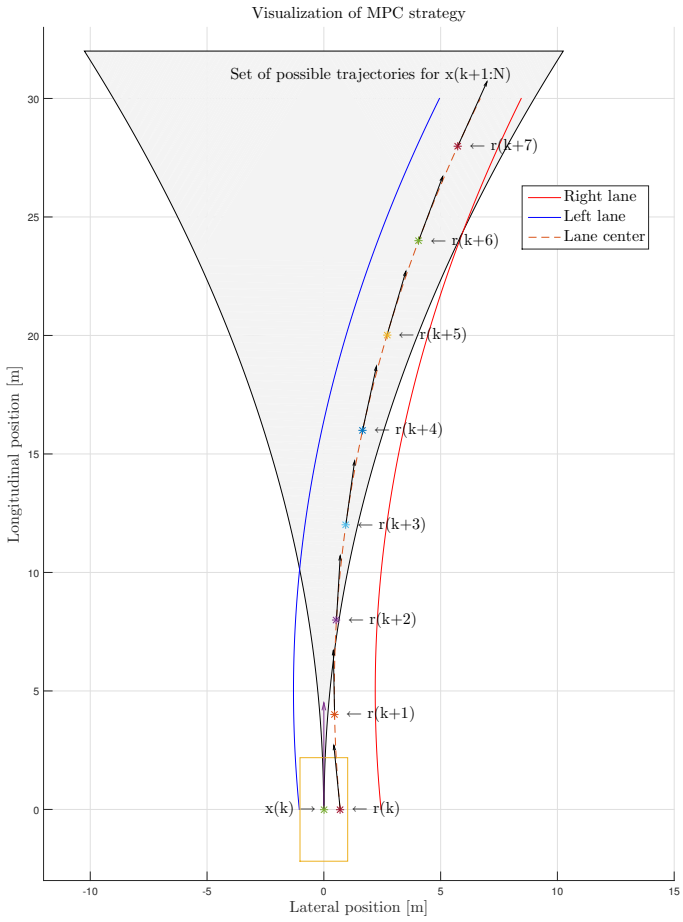


Figure 3.2: A graphical visualization of the MPC strategy.

Figure 3.2 shows the inputs to the MPC. Given a state $x(k)$, the vehicle can take any trajectory in the grey area, given a certain combination of inputs. Given the reference $r_{p_y}(k+1:k+N)$, $r_{\psi_z}(k+1:k+N)$ and $r_u(k+1:k+N)$, the MPC calculates the best combination of control signals to follow the reference as good as possible.

In this particular case, each point $r_i(k+j)$, $i\ (\in p_y,\ \psi_z)$ and $r_u(k+j)$ are given as a function of the longitudinal distance ahead as:

$$\begin{aligned} r_i(k+j) &= f_i(d(k+j)), \\ r_u(k+j) &= f_u(d(k+j)), \end{aligned} \tag{3.17}$$

where $d(k+j)$ is the longitudinal displacement in meters from the center of gravity and $f_i(\cdot)$ respectively $f_u(\cdot)$ are functions that describes the road ahead for each state and control signal.

The MPC controller uses the Mathematica generated state space representation described in (2.36). Therefore does the MPC controller have to use the output matrix (C_m) in order to convert from the Mathematica generated states to the states $(v_y, \omega_z, p_y, \psi_z)$. The rows of the output matrix corresponds to the selected states and the mapping between can be expressed as:

$$\begin{pmatrix} x_{v_y} \\ x_{\omega_z} \\ x_{p_y} \\ x_{\psi_z} \end{pmatrix} = \underbrace{\begin{pmatrix} C_{v_y} \\ C_{\omega_z} \\ C_{p_y} \\ C_{\psi_z} \end{pmatrix}}_{C_m} x_m. \tag{3.18}$$

Naturally the C_m matrix have to be inserted in the cost function defined in (3.12) and the state cost can be reformulated as:

$$\begin{aligned} &\sum_i \omega_i \left(C_i x_m(k+j) - r_i(k+j)\right)^2, \\ &i \in (v_y, \omega_z, p_y, \psi_z),\ j\ =\ 1,\ 2, \ldots, N. \end{aligned} \tag{3.19}$$

By expanding (3.19) and further separating the quadratic and linear parts of each state variable cost to:

$$\begin{aligned} &\sum_i x_m(k+j)^T \underbrace{\omega_i C_i^T C_i}_{q_i} x_m(k+j), \\ &\sum_i \underbrace{-2\omega_i r_i(k+j) C_i}_{l_i(j)} x_m(k+j), \end{aligned} \tag{3.20}$$

$$i \in (v_y, \omega_z, p_y, \psi_z),\ j\ =\ 1,\ 2, \ldots, N.$$

Then the quadratic term, q_i, for each state variable can be put in matrix form as:

$$q_x = \begin{pmatrix} q_{v_y} & 0 & \dots & 0 \\ 0 & q_{\omega_y} & \ddots & \vdots \\ \vdots & \ddots & q_{p_y} & 0 \\ 0 & \dots & 0 & q_{\psi_z} \end{pmatrix} \tag{3.21}$$

and the corresponding linear term, $l_i(j)$, can be put in vector form as:

$$l_x(j) = \begin{pmatrix} l_{v_y}(j) & l_{\omega_z}(j) & l_{p_y}(j) & l_{\psi_z}(j) \end{pmatrix}. \tag{3.22}$$

Similarly the input cost from (3.12) can also be separated in quadratic and linear parts as:

$$\begin{aligned} q_u &= w_u, \\ l_u(l) &= -2w_u r_u(k+l), \\ l &= 0,\ 1, \dots, M-1. \end{aligned} \tag{3.23}$$

The H matrix can then be constructed by putting together the quadratic parts of the cost functions. By thereafter multiplying H by 2, the H matrix is expressed in the desired standard form:

$$H = 2 \begin{pmatrix} q_x & 0 & \dots & \dots & \dots & 0 \\ 0 & \ddots & \ddots & & & \vdots \\ \vdots & \ddots & q_x & 0 & \dots & 0 \\ 0 & \dots & 0 & q_u & \ddots & \vdots \\ \vdots & & & \ddots & \ddots & 0 \\ 0 & \dots & \dots & \dots & 0 & q_u \end{pmatrix}. \tag{3.24}$$

The linear parts of the cost functions $l_x(j)$ and $l_u(l)$ can be put into the f vector in the same way:

$$f = \begin{pmatrix} l_x(1), \dots, l_x(N), l_u(0) \dots l_u(M-1) \end{pmatrix}. \tag{3.25}$$

In order to construct the equality constraint matrix A_{eq} and the corresponding b_{eq} vector. The model matrix A_m have to be calculated for the current velocity v_x whereas the input vector B_m is constant.

The state space representation is in continuous time and have to be discretized, as mentioned earlier. This is done with forward Euler method and the discrete state space can be represented as:

$$\begin{aligned} A_d &= I_n + hA_m \\ B_d &= hB_m \end{aligned} \tag{3.26}$$

The equality constraint can then be expressed as follows, with the discretized state space representation:

$$\begin{pmatrix} -I_n & 0 & \dots & \dots & \dots & 0 & B_d & 0 & \dots & 0 \\ A_d & -I_n & 0 & \dots & \dots & \dots & 0 & B_d & \ddots & \vdots \\ 0 & \ddots & \ddots & & & & & & \ddots & 0 \\ \vdots & \ddots & A_d & -I_n & 0 & \dots & \dots & \dots & 0 & B_d \\ \vdots & & \ddots & \ddots & \ddots & & & & & \vdots \\ 0 & \dots & \dots & 0 & A_d & -I_n & 0 & \dots & 0 & B_d \end{pmatrix} z = \begin{pmatrix} -A_d x_m(k) \\ 0 \\ \vdots \\ \vdots \\ \vdots \\ 0 \end{pmatrix} \tag{3.27}$$

Where n represent the number of states and h the discretization time.

The inequality constraint matrix (A_{in}) is constant and can be calculated offline but not the input vector (b_{in}) since it involves the current steering angle as can be seen in the following equation:

$$\begin{pmatrix} 0 & \dots & 0 & I & 0 & \dots & 0 \\ 0 & \dots & 0 & -I & I & \ddots & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \dots & 0 & -I & I \\ 0 & \dots & 0 & -I & 0 & \dots & 0 \\ 0 & \dots & 0 & I & -I & \ddots & \vdots \\ \vdots & & \vdots & \ddots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \dots & 0 & I & -I \\ 0 & \dots & 0 & I & 0 & \dots & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \dots & \dots & 0 & I \\ 0 & \dots & 0 & -I & 0 & \dots & 0 \\ \vdots & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & \ddots & \ddots & 0 \\ 0 & \dots & 0 & \dots & \dots & 0 & -I \end{pmatrix} z \geq \begin{pmatrix} \dot{u}_{min} T_{MPC} + \delta_f(k) \\ \dot{u}_{min} h \\ \vdots \\ \dot{u}_{min} h \\ \dot{u}_{max} T_{MPC} - \delta_f(k) \\ \dot{u}_{max} h \\ \vdots \\ \dot{u}_{max} h \\ u_{min} \\ \vdots \\ \vdots \\ u_{min} \\ u_{max} \\ \vdots \\ \vdots \\ u_{max} \end{pmatrix} \tag{3.28}$$

where the deviation constraint is reformulated to a rate constraint times a sampling time as $\Delta u_{max} = \dot{u}_{max} h$. The lower and upper bound can be formulated as u_{min} respectively u_{max}.

3.2.2 Tuning the MPC

All state variables and the input signal must have a weight $w_i > 0$ because the quadratic optimization matrix H is required to be positive definite for some solvers,

[14]. For lateral position, yaw angle and the input signal, the weighting determines the importance of minimizing the deviation from the reference. Whereas for lateral velocity and yaw rate the weights penalize deviation from zero.

The control and prediction horizon affects the performance of the MPC. By increasing the horizons the optimization problem becomes larger and more computational demanding to solve.

The control and prediction horizons also affects the behavior of the controller. When the control horizon tend to be smaller than the prediction horizon, the controller is forced to regulate faster and harder. A small control horizon makes the vehicle turn earlier into a corner than a controller with long control horizon. On a straight road, a short control horizon increases steady state oscillations on the steering wheel angle. In order to trade of between these facts the control M and prediction N horizon have been selected as:

$$\begin{aligned} N &= 10, \\ M &= 4. \end{aligned} \tag{3.29}$$

The steering angle and the steering angle rate constraints are important since these limits the controller to not violate the linear mathematical model and making unreasonable steering angle request. Similar to [2], the constraints presented in the following table have been used:

$$\begin{aligned} u_{max} &= 30^\circ, \\ u_{min} &= -30^\circ, \\ \dot{u}_{max} &= 20^\circ/s, \\ \dot{u}_{min} &= -20^\circ/s. \end{aligned} \tag{3.30}$$

The discretization time in combination with the number of predictions is determining the distance between each prediction point and hence the longitudinal prediction range in meters ahead of the vehicle. The discretization time have been fixed to:

$$h = 0.09\,\text{s}. \tag{3.31}$$

which make the prediction range varying according to the longitudinal velocity of the vehicle.

3.3 Finite Horizon Linear Quadratic Controller

Another approach to controlling the system with model prediction is to remove all constraints and instead punish certain signals with higher costs. This is called a Finite Horizon Linear Quadratic Controller (FHLQC). Over a finite prediction

horizon, a cost function is defined as the sum of the weighted square errors. This problem is much more computationally efficient, since the control signal is calculated by a single matrix multiplication. The computational time is therefore deterministic.

A few words on quadratic optimization on matrix form. A quadratic objective function without constraints is be defined as:

$$\min_x \quad J = \mathbf{u}^T\mathbf{H}\mathbf{u} + \mathbf{u}^T\mathbf{f}, \tag{3.32}$$

where $\mathbf{u}$ is the optimization vector, and $\mathbf{H}$ and $\mathbf{f}$ are constant matrices that define the problem. The optimal solution of this function, i.e. the the lowest cost, is given by setting its derivative with respect to $\mathbf{u}$ equal to zero. Taking the derivative:

$$\frac{\partial J}{\partial \mathbf{u}} = \mathbf{u}^T(\mathbf{H}^T + \mathbf{H}) + \mathbf{f} = 0. \tag{3.33}$$

The optimal solution is therefore given by:

$$\mathbf{u}^* = -(\mathbf{H}^T + \mathbf{H})^{-1}\mathbf{f}. \tag{3.34}$$

Now to define the control problem in quadratic form. For a discrete state space system:

$$x(k+1) = Ax(k) + Bu(k), \tag{3.35}$$

any future state $x(k+K)$ can be calculated with the initial state and the control signals up to this time instance:

$$x(k+K) = A^K x(k) + \sum_{i=0}^{K-1} A^{K-1-i}Bu(k+i). \tag{3.36}$$

We define $e(k+n) = x(k+n) - r(k+n)$ to preserve space in calculations performed later on. Putting up the same cost function as for the MPC:

$$J = \sum_{n=1}^{N} e(k+n)^T Q e(k+n). \tag{3.37}$$

Substituting $x(k+n)$ with $A^n x(k) + \sum_{i=0}^{n-1} A^{n-1-i}Bu(k+i)$ give:

$$e(k+n) = \underbrace{A^n x(k)}_{\alpha_n} + \underbrace{\sum_{i=0}^{n-1} A^{n-1-i}Bu(k+i)}_{\beta_n} - \underbrace{r(k+n)}_{r_n}. \tag{3.38}$$

The cost function then becomes:

$$L = \sum_{n=1}^{N} \left(\alpha_n + \beta_n - r_n\right)^T Q \left(\alpha_n + \beta_n - r_n\right). \tag{3.39}$$

Only the terms with a β_n-factor, which contain $u(k+i)$, will remain after the partial differentiation. Therefore, any term without a β_n-factor will grouped into a constant term c_n.

$$L = \sum_{n=1}^{N} \left(\alpha_n^T Q \beta_n + \beta_n^T Q \alpha_n - r_n^T Q \beta_n - \beta_n^T Q r_n + \beta_n^T Q \beta_n + c_n\right). \tag{3.40}$$

Utilizing the fact that all off diagonal elements of Q are zero, the transposed terms can be added together.

$$L = \sum_{n=1}^{N} \left(2\alpha_n^T Q \beta_n - 2r_n^T Q \beta_n + \beta_n^T Q \beta_n + c_n\right). \tag{3.41}$$

The terms with one β_n-factor are the first degree terms, and the $\beta_n^T Q \beta_n$-term is the second order term. In equation (3.32), the $\mathbf{f}$ vector is multiplied with the first order term, and the $\mathbf{H}$ matrix is a factor in the second order term. Therefore, the factors multiplied with $u(k+i)$ will construct the $\mathbf{f}_u$ vector, while factors which multiply with $u(k+i)u(k+j)$ will construct the $\mathbf{H}_u$ matrix.

The $\mathbf{f}_u$ vector:

$$\mathbf{f}_u : 2\sum_{n=1}^{N} \left((A^n x(k))^T Q \sum_{i=0}^{n-1} A^{n-1-i} B u(k+i) - \left(\sum_{i=0}^{n-1} A^{n-1-i} B u(k+i) \right)^T Q r(k+n) \right). \tag{3.42}$$

The $\mathbf{f}_u$ vector will be constructed by iterating n from 1 to N. For each $i = 0 : n-1$, the corresponding element $\mathbf{f}_u(i)$ will be the sum of all the terms that are $(A^n x(k))^T Q A^{n-1-i} B - (A^{n-1-i} B)^T Q r(k+n)$.

The $\mathbf{H}_u$ matrix:

$$\mathbf{H}_u : \sum_{n=1}^{N} \left(\sum_{i=0}^{n-1} A^{n-1-i} B u(k+i) \right)^T Q \sum_{j=0}^{n-1} A^{n-1-j} B u(k+j). \tag{3.43}$$

In this particular problem, the control signal dimension is 1×1, which allows us to move the control signal factors without compromising the matrix multiplications. Moving the control signal factors and rewriting the summations give:

$$\mathbf{H}_u : \sum_{n=1}^{N} \sum_{i=0}^{n-1} \sum_{j=0}^{n-1} B^T (A^{n-1-i})^T Q A^{n-1-j} B u(k+j) u(k+i). \tag{3.44}$$

The $\mathbf{H}_u$ matrix will be constructed by iterating n from 1 to N. For each $i = 0 : n-1$ and $j = 0 : n-1$, the corresponding element $\mathbf{H}_u(i,j)$ will be the sum of all the terms $B^T(A^{n-1-i})^T Q A^{n-1-j} B$.

The cost function is a quadratic function, so if we can identify the corresponding $\mathbf{H}$ matrix and $\mathbf{f}$ vector, and setting $\mathbf{u} = [u(k)\ u(k+1)\ u(k+2)\ ...\ u(k+N)]^T$, we can get the optimal control vector $\mathbf{u}^*$ from:

$$\mathbf{u}^* = (\mathbf{H}_u^T + \mathbf{H}_u)^{-1}\mathbf{f}_u. \tag{3.45}$$

3.3.1 Implementation of the FHLQC

The FHLQC uses the linear model derived in section 2, in which there are four states: lateral velocity, yaw rate, lateral offset and angular offset. It also includes the EPAS dynamics as a fifth state, such that the controller will take this into account. Instead of multiplying the transfer function for the vehicle with the transfer function for the steering wheel, the steering wheel angle is included as a fifth state in the state space vector, as described in section 2.3.

The $\mathbf{H}_u$ matrix only depends on v_x, which affects the discretization, and Q, which affects the weighting. The $\mathbf{f}_u$ vector has to be calculated for each new sampling instance, since it depends on the current state and future reference.

Each control signal $u(k+n)$, where $n = 0,\ 1\ , ..., N-1$, is calculated by the vector multiplication of the n:th row of $(\mathbf{H}_u^T + \mathbf{H}_u)^{-1}$ with $\mathbf{f}_u$. If the controller can be tuned in the test phase, with ideal parameters for all velocities, the controller could be implemented with look up tables for each input signal to the controller. The control signal is a linear combination of the states and the sampled reference points, where each state and sampled reference is weighted with its own coefficient.

$$\begin{aligned} u(k) = v_y(k)h_{v_y}(v_x)\ +\ \omega_z(k)h_{\omega_z}(v_x)\ +\ p_y(k)h_{p_y}(v_x)\ +\ \psi_z(k)h_{\psi_z}(v_x) + \\ \delta(k)h_\delta(v_x)\ +\ r(k+1)h_{r(k+1)}(v_x)\ +\ \ldots\ +\ r(k+N)h_{r(k+N)}(v_x). \end{aligned} \tag{3.46}$$

In (3.46), each state and sampled reference is weighted with a weight that depends on the longitudinal velocity. As such, the FHLQC is computationally light.

4

Real world test system

The real world tests system consists of a number of different components working together. A vision system senses and estimates the lane markings relative to the car. These estimations are then transmitted over CAN to a computer system, which reads the data via CAN. The computer system also receives data about the car state such as yaw rate, longitudinal/lateral velocity and current steering wheel angle. All of this data is fed to a control algorithm, which calculates the desired steering wheel angle. This desired steering angle is sent over the CAN bus as a request to the steering servo.

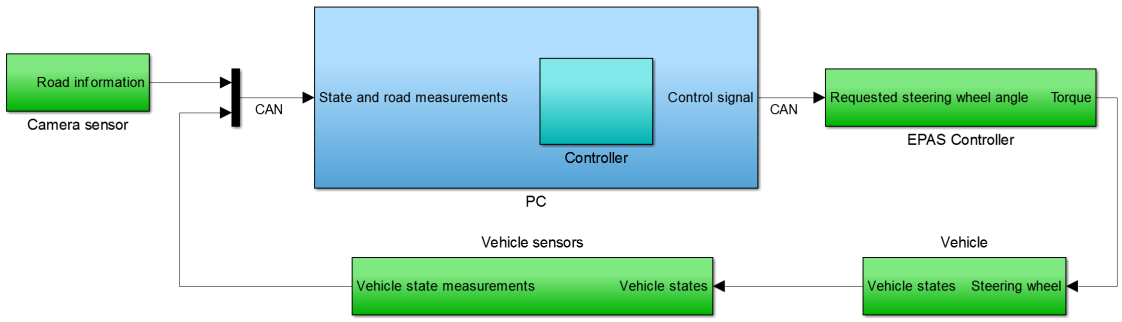


Figure 4.1: This is the output from the camera and the vision algorithm. The lane estimations are shown with dashed green lines in the image.

4.1 Test vehicle setup

The car used for testing is a Volvo V40, manufactured 2012. It has a camera mounted behind the windshield and a radar mounted in the grille. These systems are developed and manufactured by Delphi. The car is also equipped with other sensors such as gyroscope, speedometer and accelerometer that are used in the control algorithm. All the relevant signals are available over the CAN network.

4.2 Steering

The car has an electric power assisted steering (EPAS) system which is available via CAN. With a certain CAN signal, the steering wheel angle, and thus the wheel angle,

can be controlled via the CAN bus. The EPAS system has an internal controller which uses torque from a servo motor, connected to the steering wheel, to achieve the requested steering wheel angle. The steering servo can achieve torque high enough to maneuver the car at low speed, which is used for automated parking. At speeds over 10 km/h, the high torque capabilities are disabled internally in the EPAS node. At 65 km/h the EPAS is again enabled, but only with torque low enough to be countered by the driver by simply holding the steering wheel. As such, the driver can intervene if he or she feels uncomfortable with the requested steering wheel angle. The steering wheel behaves like a rate constrained first order system. For low angular velocities of the steering wheel, the dynamics is linear. The absolute rate is however limited.

4.3 Camera sensor

A camera sensor is used for lane marker detection and estimation. It uses specialized hardware that runs the algorithms efficiently enough to be executed in real time. The vision algorithms detects and estimates which pixels that belongs to the lane markings. These pixels are then used to fit polynomials that can represent the lane markings.

4.4 Control system hardware

The control system is run on a standard laptop PC with a 2.9 GHz Intel i7 processor and 8 GB of RAM. Using a Vector CANcaseXL unit, the control system software receives information about the car state, and sends angle requests to the steering wheel controller.

4.5 Control system software

The control system software is written in C++ and built using Microsoft Visual Studio. Using API from Vector Informatik, the application can communicate with a CANcaseXL unit, which reads signals from the vehicle and the camera, and sends out steering wheel angle requests to the EPAS controller. The entire application is spread over several threads; the CAN receive thread, the control system thread and the GUI thread. The CAN receive thread runs every 2 ms in order to capture all the data.

4.6 Lane marker estimation

The camera senses the left and right lane in the form of a third degree polynomial. There is also a range associated with the polynomial, which represents how far the

lane is estimated, in the longitudinal direction of the vehicle. The polynomial has the following form:

$$f(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3, x \in [0, x_{max}]. \tag{4.1}$$

The first coefficient, a_0, determines the lateral position of the lanes relative to the car. The second coefficient, a_1, determines the relative angle between the car heading and the lanes. The third and fourth coefficient, a_2 and a_3, represent the curvature of the road. x_{max} is the range of the current estimate; beyond this point the algorithm is not certain of the line characteristics. The center offset is simply the average between the right and the left coefficients:

$$a_{c0} = (a_{r0} + a_{l0})/2. \tag{4.2}$$

The rest of the coefficients are the same, only the lateral offset is estimated individually.

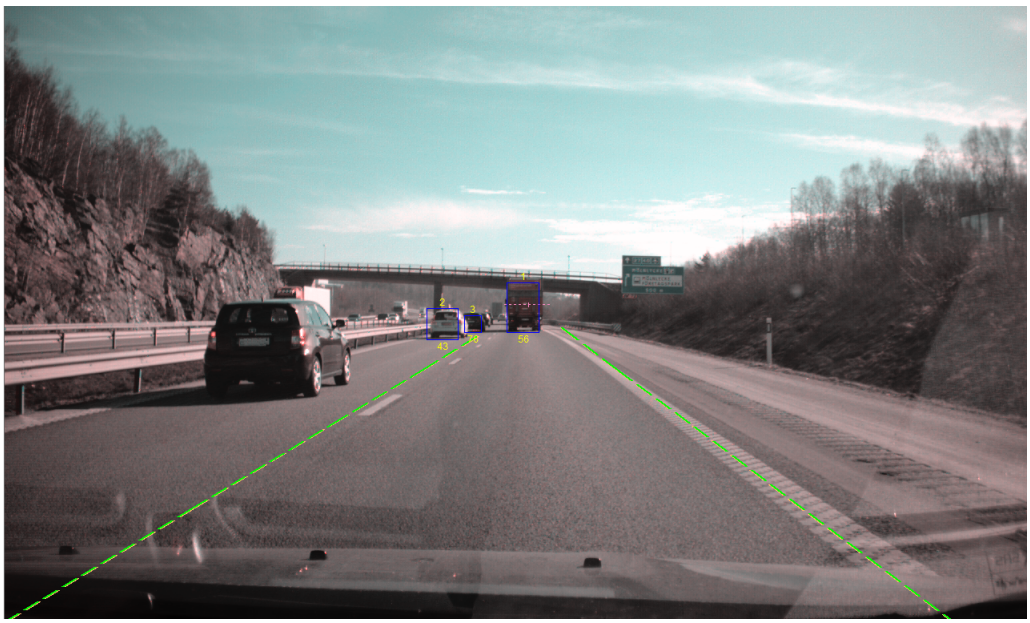


Figure 4.2: This is the output from the camera and the vision algorithm. The lane estimations are shown with dashed green lines in the image.

In figure 4.2 it can be seen how the vision algorithm estimates the lane markers. When the vision algorithm makes estimates of the lane markers, they are done in the perspective of the camera, i.e. projected from road plane onto the image sensor plane. Mounting parameters such as the linear and angular offset relative to the vehicle coordinate system are known to the camera. Thus, assuming that the road below is relatively flat, the detected lines can then be unskewed and transformed to the vehicle coordinate system.

5

Simulation

Simulations have a lot of benefits for rapid development that is not possible or harder to achieve in real world tests. For example can simulations run faster than real time, specific test scenarios can be constructed easily and measurements can be ideal.

Therefore have simulation been mainly used for validating the mathematical model and developing controllers instead of committing real world tests. Since the simulations is a way to substitute real world test is the accuracy of the simulated dynamics important so that the simulation is as realistic as possible.

5.1 Simulation platform

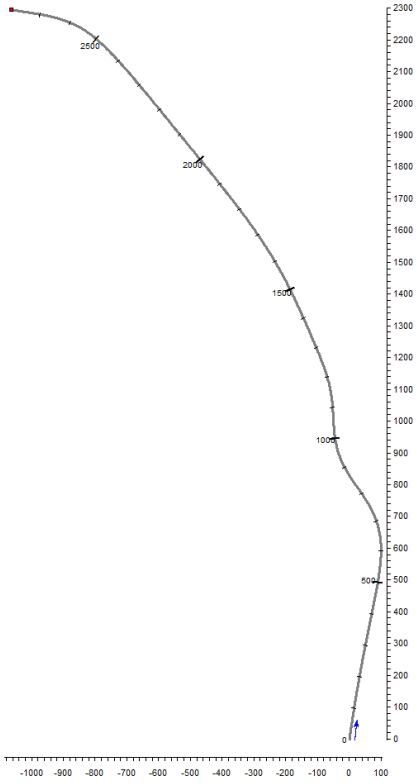
IPG Automotive is the provider of the simulation software CarMaker, in which a lot of time have been spent on adapting the dynamics of real vehicles. The nonlinear dynamics of the wheels and the slip are included, as well as vertical dynamics from the dampers. One big advantage with CarMaker is that it can be used together with Matlab/Simulink such that the vehicle dynamics are simulated in CarMaker, and control system calculations are done in Simulink. This makes it possible to run the controller in closed loop with the simulated vehicle.

5.1.1 Test track

The test tracks in CarMaker can either be constructed manually by adding straight and curved roads, or extracted from real data from Google Maps. This allows for simulations that accurately represent scenarios that the real vehicle will be tested in.

For the simulation, a segment of the E6 motor way in Gothenburg have been selected to be implemented in CarMaker. This road segment have both tight corners with high curvature and a longer corner with smaller curvature and is one of the most changeling scenarios in a close radius of Gothenburg. The road segment can be seen in figure 5.1 and will further be referred as R_1.

(a) Satellite image from Google Maps.



(b) Bird's eye view of the road segment implemented in CarMaker.

Figure 5.1: This figure shows that the real world road segment can be inserted from Google Maps into Carmaker.

5.1.2 Implementation

There are several options for vehicle sensors in CarMaker. Many of the states are available directly through blocks, such as yaw rate, global position, velocity. The simulation implementation is partly done in Simulink as can be seen in figure 5.2. The five bigger blocks represent the main task of the simulation and the arrows also show the work flow of the simulation.

In order to mimic the real test system, a line sensor was added. The line sensor is described as a perfect vision sensor, which detects the road marking ahead of the vehicle. The measurements are then given in the form of a number of global x- and y-coordinates, which represents points on the lane marker. The points are then translated to the origin, and then rotated to the same orientation as the vehicle. With this translation and rotation, the points are now in the vehicle coordinate frame.

Since the output from the vision algorithm in the actual system gives the lane estimation as a third degree polynomial, the points from the line sensors are fitted into such a polynomial as well in the *Lane marking decoder* block, even though it is the actual points that are used in the controller algorithm.

The lane marker polynomials are then fed to the reference generator which calculates the center line polynomial among other parameters that can be used as references for the controller. The MPC controller uses the curvature as input signal where as the FHLQC does not use it. This means that the reference generator block can be constructed according to what references the controllers uses. The idea is that the input to the reference generator is the same for both the simulation and the real test vehicle.

The controller algorithm is then fed with the reference trajectory parameters and the current steering wheel angle, yaw rate, longitudinal and lateral velocity. The output of the controller is the new steering angle that is fed forward as a requested steering wheel angle to the EPAS system.

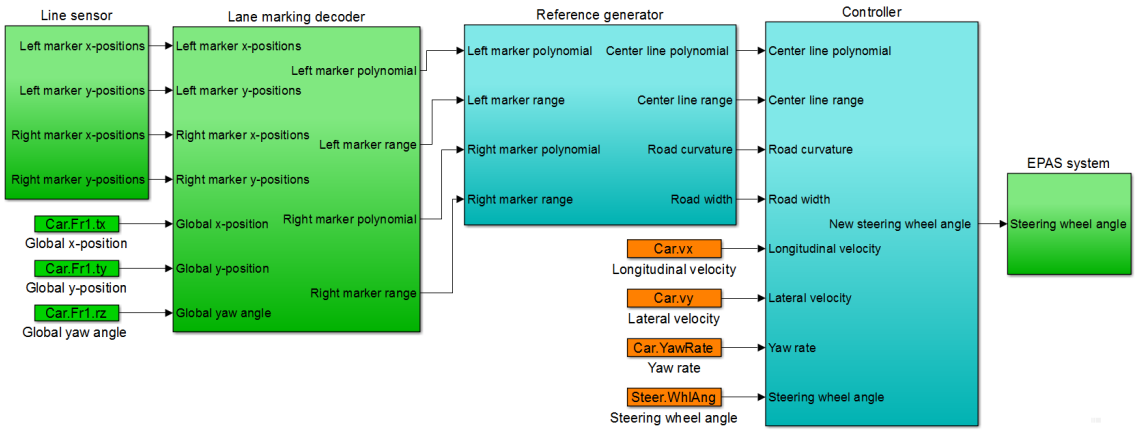


Figure 5.2: This figure shows the Simulink implementation of the different parts. The cyan colored blocks shows the reference generator and the controller. These are the same for both simulation and real test. The green colored blocks is specific for the simulation implementation and the orange colored blocks represent the input signals to the controller.

5.1.3 Test vehicle

The vehicle in the simulation is desired to be the same as the real test vehicle so that the mathematical model and controllers can be used in both without any or only just small modifications. CarMaker have several example vehicles to chose among but unfortunately this thesis, no Volvo V40.

The example vehicle in CarMaker which is most similar to the Volvo V40 is a Ford Focus manufactured 1998. Since this vehicle have approximately the same size and weight as the Volvo V40, the dynamics are assumed to be similar between these two vehicles.

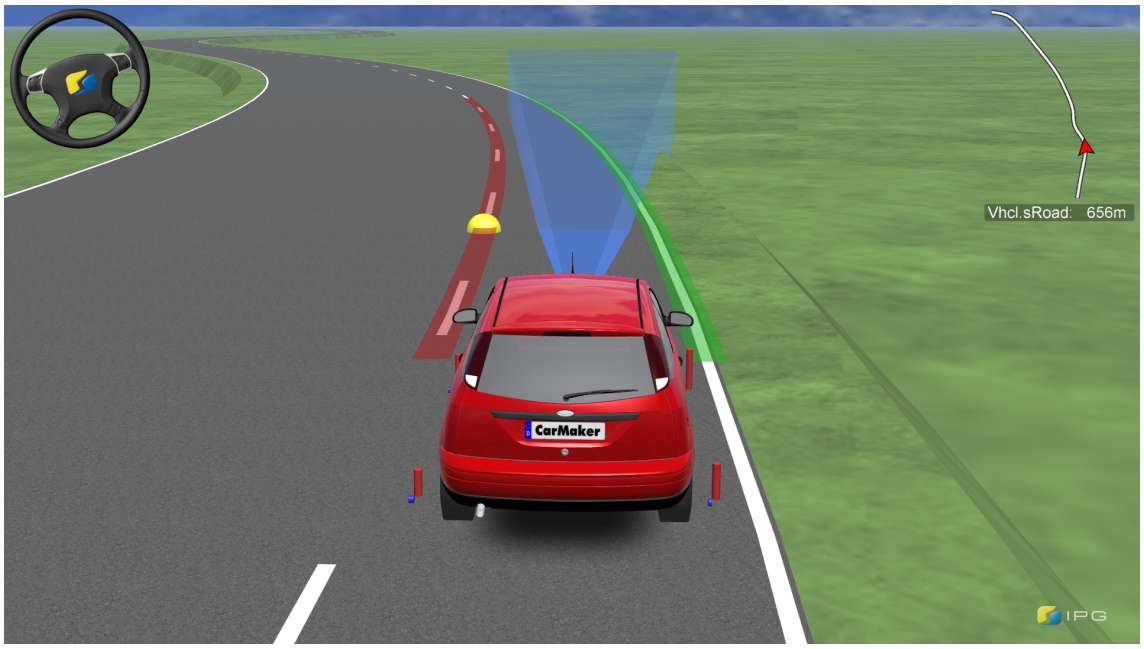


Figure 5.3: This figure shows the simulation environment. The lane marker estimations are shown as a red and green lines. The steering wheel in the upper left corner shows the current steering wheel angle and the test track, R_1, can be seen in the upper right corner.

5.2 Model Parameter Estimation & Development

The parameters of the mathematical model needs to be correct for both simulation and vehicle implementation. Many of the vehicle parameters can be found by reading about the Ford Focus model in CarMaker. For example, the position of COG, the weight, the length between wheel axis and COG and the body inertia which can be found in 5.1.

m	$1174\,\mathrm{kg}$
L	$2.680\,\mathrm{m}$
l_f	$1.0430\,\mathrm{m}$
l_r	$1.6370\,\mathrm{m}$
I_{zz}	$1720\,\mathrm{kg}\cdot\mathrm{m}^2$
δ_{ratio}	15.5

Table 5.1: Mathematical Model parameters

Carmaker is using a more advanced approach for modeling tire dynamic and therefore can the cornering stiffness parameters c_0 and c_1 not be directly determined from the Ford Focus data sheet. Therefore must these parameters be estimated instead. This was done by formulating an optimization problem where the cost function is to minimize the error between the state variables of the mathematical model output and the CarMaker model output when controlled by the same steering

angle sequence. The variables to optimize are the cornering parameters which are constrained to be constant for the entire optimization.

The optimization was preformed for different longitudinal velocities and as can be seen in Table 5.2. The values are changing for different velocities even though the same steering angle sequence was used in all test. However, the variation is small and the cornering stiffness parameters can be approximated to be constant. The MPC implements that the parameters varying as estimated but the FHLQC on the other hand uses constant values.

c_0	38.016	39.724	40.432	40.312	39.753	39.113
c_1	-0.0074	-0.0075	-0.0075	-0.0072	-0.0069	-0.0066
v_x	70 km/h	80 km/h	90 km/h	100 km/h	110 km/h	120 km/h

Table 5.2: Estimated tire cornering stiffness parameters. The parameters are depending on the longitudinal velocity of the vehicle.

6

Results

This chapter gathers results from all previous chapters in a chronological order. First the mathematical model performance and discetization stability analysis will be presented from the Modeling Chapter. Followed by results from simulation and real world tests of the two controller algorithms.

6.1 Model validation

In order to ensure the accuracy of the linear model, when comparing it to the nonlinear model in CarMaker, model validation is used. The systems are tested open loop, such that a comparison can be made between the outputs from the different models given the same input. Two types of inputs are tested; a discontinuous step with a certain amplitude, and a zero mean Gaussian noise with a certain variance. The longitudinal velocity was set to 100 km/h during the tests. Both CarMaker and the linear model use the same input from the steering wheel, so studying that signal when comparing the two models is not of interest.

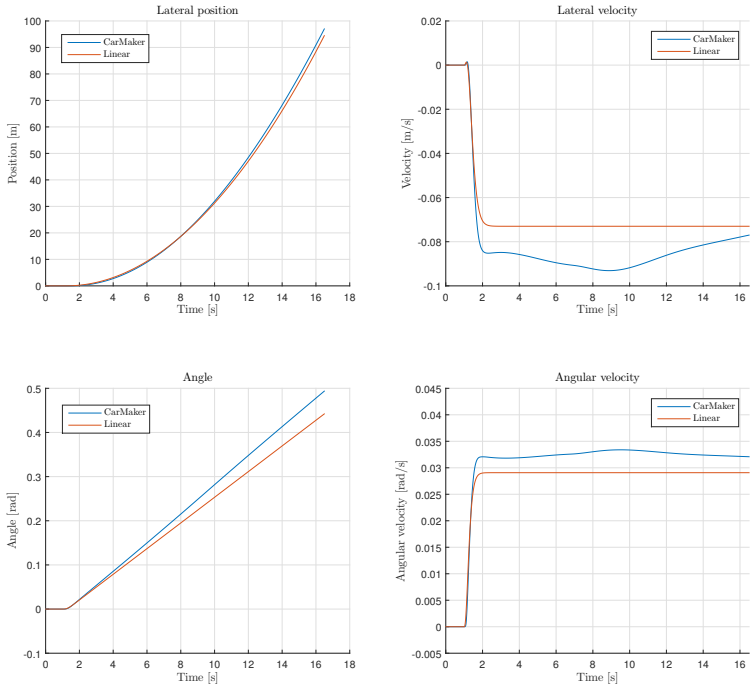


Figure 6.1: These figures show the response for four of the different states when the input signal is a discontinuous step with and amplitude of 0.1 rad.

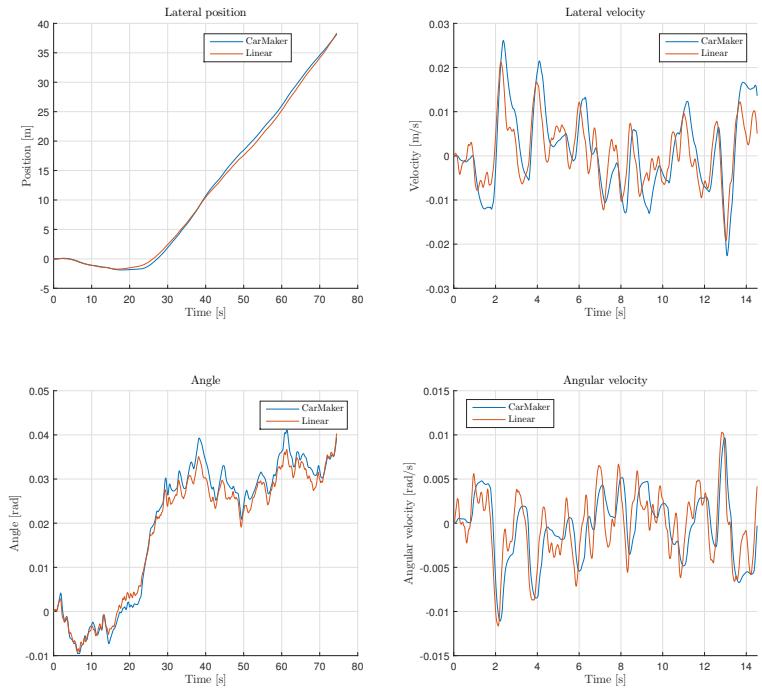


Figure 6.2: In this set of figures the input is random Gaussian noise with a variance of 0.01 rad and a sampling time of 0.01 s.

The figures show that the linear model has the same characteristics as the model that CarMaker uses. In the step responses, the difference between how the models give substantially different results showthat there are unmodeled dynamics as well as uncertainty in the model parameters.

Figure 6.1 shows that there is a difference in steady state gain between the linear model and CarMaker. The bumps in the lateral velocity and the yaw rate, after the transient, shows that there is some dynamics that the linear model does not fully capture. Over time however, the lateral position and the angle does not differ more than a few meters and a few degrees respectively.

When the input signal is random noise with more high frequency content, shown in Figure 6.2, the correlation between the different models is better. The lateral position error and the angle error is significantly smaller compared to the step response, despite the fact that this test lasted about 75 s, whereas the step response lasted about 17 s. The lateral velocity and the angular velocity can also be seen to show the same characteristics.

There are more figures showing different step amplitude and another noise variance for the input in Appendix B.2.

6.1.1 Dynamics of the EPAS system

The closed loop steering system has been observed to behave approximately as a first order system with a time constant of about 0.4 s as can be seen in Figure 6.3. This shows that there exist dynamics between the requested steering wheel angle and the actual steering wheel angle which neither the mathematical model or the simulation adapts without modifications.

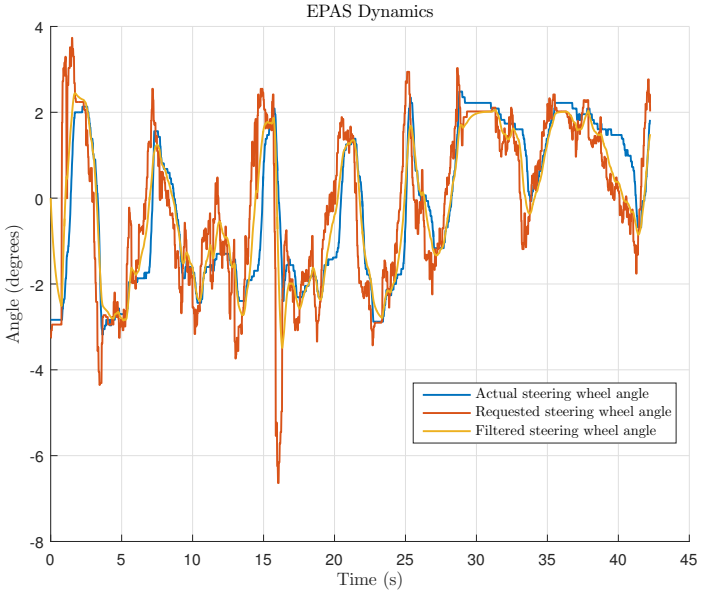
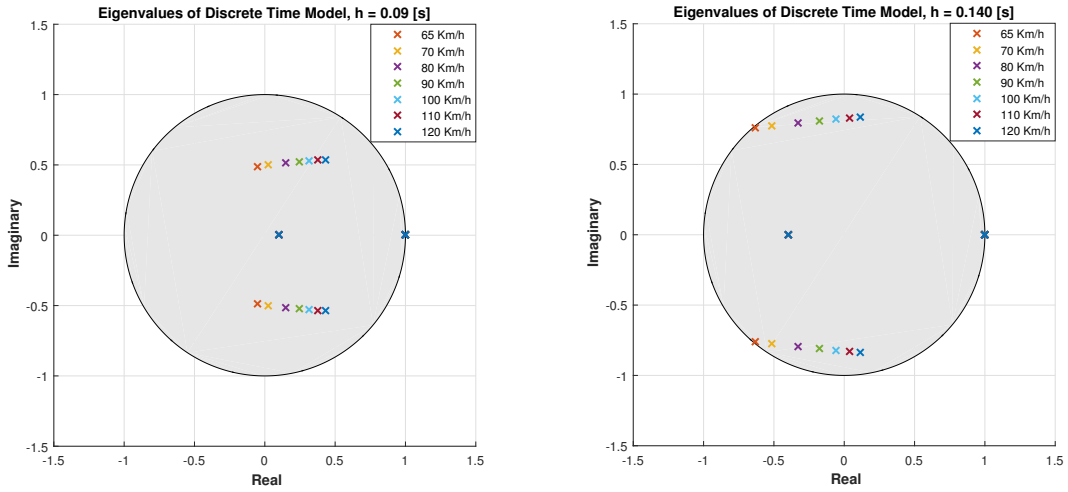


Figure 6.3: As can be seen in the figure, the requested steering angle in red deviates from the actual steering angle in blue. The filtered steering wheel angle in yellow is fitting the actual steering angle better than the requested steering angle.

6.2 Discretization Stability

The discretization step directly affecting the stability of the discrete time model as described in 2.4 and the stability criterion is limiting the discretization step to be lower than a upper bound which is determined by the eigenvalues of the continues time model. Since the discretization step is affecting the distance that the vehicle travels between each sample, a small discretization step will require more samples to get the same range in meters than a greater discretiztion step. The prediction horizon of the MPC needs to have a long enough look a head range and therefore is the discretization step also affecting the number of prediction samples that is needed which affects the computation time of the optimization.

Since the model dynamics is depending on the longitudinal velocity will the eigenvalues and hence the stability of the discretization change for different longitudinal velocities. This can be seen in the Figure 6.4.



(a) Smaller discretization step increase stability margins.

(b) Discretization is marginal stable for $h = 0.140$ and longitudinal velocity $v_x = 65km/h$.

Figure 6.4: The eigenvalues depends on the longitudinal velocity and the discretization step. All eigenvalues fulfill $|1 + h\lambda| < 1$ and the discretization is stable for velocities of interest.

Another aspect to take into account when selecting discretizaton step is that larger step sizes increases the truncation error between the true state and the approximated discretized state, [6]. Much due to the truncation error, the discretization time $h = 0.09$ have been used for the MPC controller.

6.3 Test track in simulation

The two controllers were tested on test track R_1 for three different velocities; 70 km/h, 100 km/h and 120 km/h. The only test case shown here is that for 120 km/h, since the other test cases had similar results. The results for the other velocities are shown in appendix. The figures show the performance of the built in controller from CarMaker, the MPC and the FHLQC.

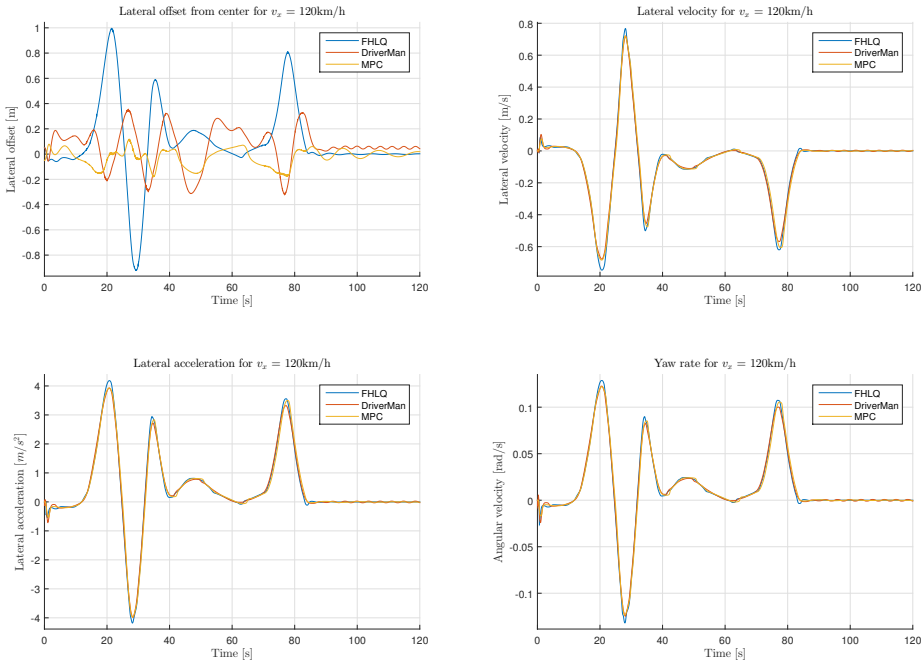


Figure 6.5: These plots show four different performance factors when the longitudinal velocity is 120 km/h

In Figure 6.5, the lateral offset, velocity and acceleration, as well as yaw rate are shown in comparison for the two different controllers and CarMaker driver. For the velocity, acceleration and yaw rate, only minor differences can be distinguished. The only noticeable detail that can be said anything about is that the FHLQC has a peek velocity, acceleration and yaw rate that is larger than the MPC and the CarMaker controller. It is in the sub figure showing the lateral velocity where things differs. Both the MPC and the FHLQC are tuned such that they have minimal oscillations on straight roads. This tuning also affects the behaviour when the road is turning. The FHLQC can be seen to deviate from the lane center more than the other controllers. This is difficult to see, but in right turning curves it keeps to the left and in left turning curves it keeps to the right. The MPC and CarMaker controller on the other hand, cut corners and deviate from the lane center less than the FHLQC.

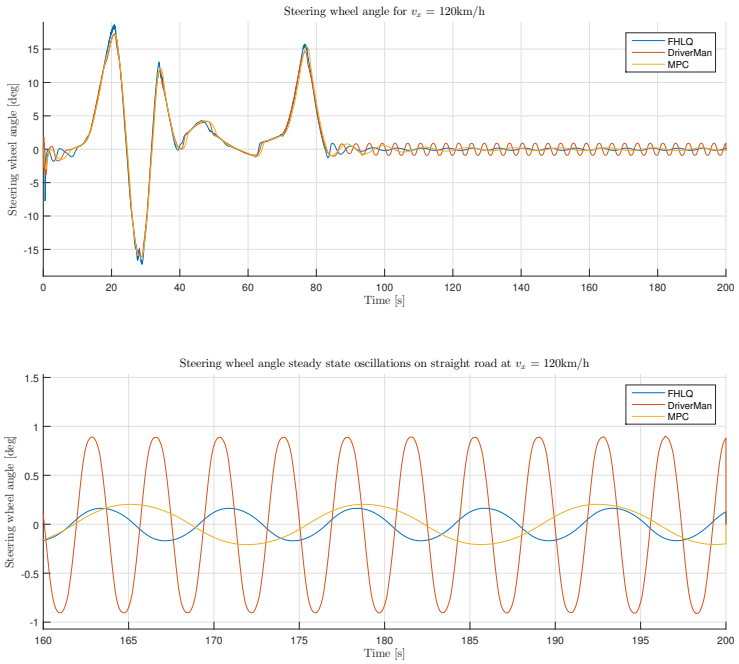


Figure 6.6: The steering wheel angle for when the longitudinal velocity is 120 km/h. The lower plot is a zoomed in version of the first one, to better visualize the steady state oscillations.

Figure 6.6 shows the steering wheel angle for the same simulations as in Figure 6.5. During the first part of the track, it is difficult to see any particular differences to draw conclusions from. When the car is driving on the straight road however, after 80 s, the steady state oscillations are visible. The CarMaker controller has a larger peak to peak angle and a higher frequency in the oscillations. The MPC and the FHLQC however, have smaller peak to peak amplitude and lower frequency in their oscillations. The values of peak to peak amplitude and frequency can be seen in Table 6.1.

Controller	Peak to peak amplitude (degrees)	Frequency (Hz)
MPC	0.4106	0.1471
FHLQC	0.3319	0.2632
CarMaker	1.791	0.5263

Table 6.1: Steady state steering wheel oscillations.

6.4 Real world tests

These tests were carried out on highways when driving at 100 km/h. Due to time limitations, the different controllers and the human driver were not driven on the

same road segment. These results can therefore only give a general indication of the performance. The roads segments are however considered to be similar enough to draw some conclusions from.

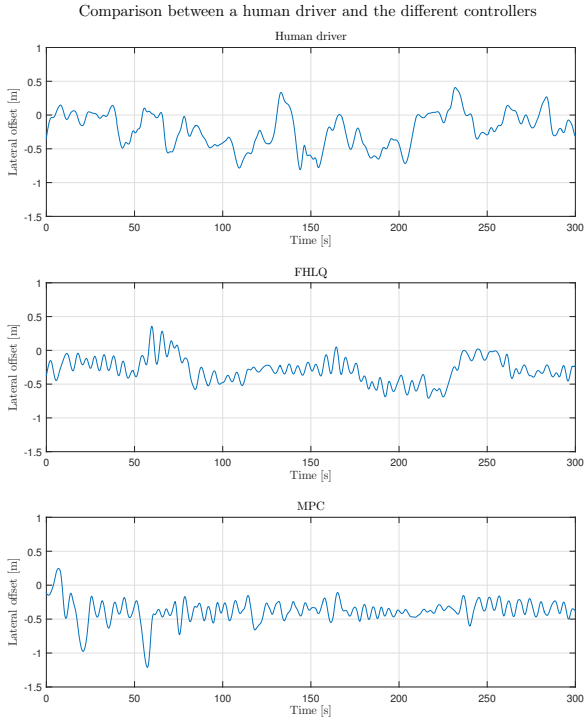
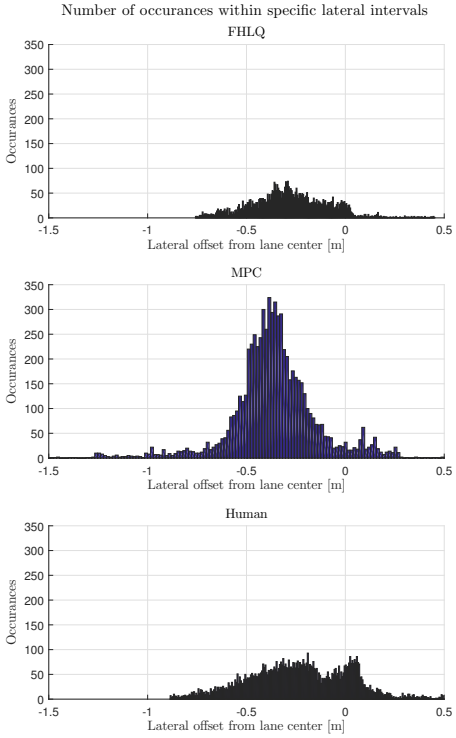


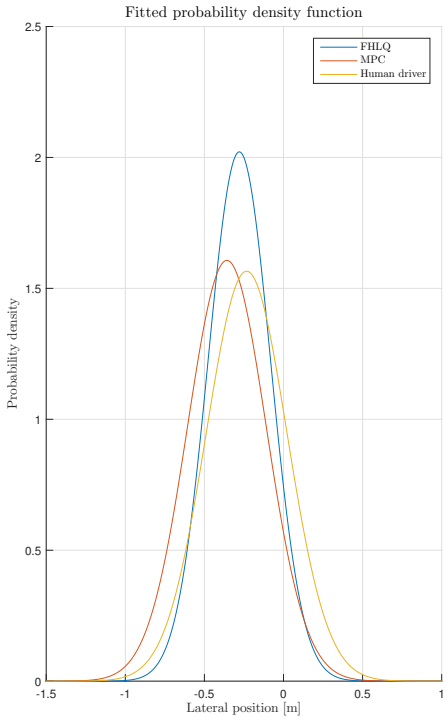
Figure 6.7: The lateral offset is post processed to remove high frequency noise from the sensors readings.

Figure 6.7 shows a comparison lateral offset over time for the FHLQ, the MPC and the human driver. The two controllers show similar behaviour in terms of oscillation frequency and amplitude.

In Figure 6.8a, it can be seen that the controllers are better than the real driver on staying in the center of the road since the lateral offset distribution is more flat for the human driver. The FHLQC have the most narrow distribution which is the reason to why the peak of density function is higher for the FHLQC in Figure 6.8b.



(a) This figure shows histograms of the lateral offsets for the controllers and human driver. As can be seen the human driver have a wider and flatter distribution than the controllers. It can also be seen that the MPC have the larger tails than the FHLQC.



(b) This figure shows the probabilistic density function for the controllers and human driver. The distribution is more narrow for the FHLQC controller in comparison with the MPC and human driver.

Figure 6.8: This figures shows the lateral offset between vehicle center and road center for the controllers and the human driver in terms of distributions.

6.5 Execution time tests

An interesting factor to test is the execution time of the different controllers. The MPC had an average execution time of 10 ms when run on the project laptop, described in Section 4.4. The prediction horizon was set to 10 and the control horizon was set to 6. The FHLQC had an execution time of about $50\mu s$, with a prediction and control horizon of 20. Another test was also carried out to test the MPC performance on an embedded system. The embedded system used a 32 bit ARM processor from the STM32 family, with a hardware floating point unit running at a clock frequency of 168 MHz. This test did not run the actual MPC routines on the embedded system, but did instead a crude test to examine the performance. Instead of running the MPC routines, the test performed the same number of floating point operations as the MPC routines would have done, to give

an indication of the time consumption. Running only the floating point operations, one sampling instance was completed in 130 ms. The aimed for sampling time of the MPC as of now is 10 ms. The FHLQC was not tested on the embedded system, due to lack of time.

7

Conclusion

Results from the simulation shows that the mathematical model is good enough to be used to predict the future states. Both the MPC and FHLQC use this strategy for calculating the controller output. The controllers show promising results in simulation with quite similar performance. In comparison, the MPC seems to have lower frequency of the steady state oscillation when the reference trajectory is straight and deviates less from the center line. The MPC also cuts corners better, which is a desirable behaviour.

A clear disadvantage for the MPC is that it is computationally heavy, and it is therefore probably not feasible to implement the controller in existing automotive hardware systems, where the performance is significantly lower than on a PC. The execution time of the FHLQC is however not considered to be computationally demanding, relative to the size of its task. From the test results, it is clear that the MPC is not capable of running in real time on an embedded system.

The FHLQC does not model feed forward for the steering wheel angle or lateral velocity, or look up tables for model parameters. Since its performance is equal to that of the MPC, such detailed modeling might be unnecessary.

A clear performance issue was the oscillations that were present when the real world tests were performed. This occurs due to the inaccuracy and poor resolution of the EPAS controller. It is clear that a better accuracy is needed to achieve oscillation free lateral control.

Regarding what type of controller is best suited for this type of problem is still difficult to answer. Both the controllers achieve similar performance, with the MPC being much more computationally demanding. If further control design and testing would be performed, the MPC could probably outperform the FHLQC. From the results in this report however, the conclusion to draw is that the FHLQC is probably better than the MPC due to its similar performance in controlling the vehicle, while still not demanding any substantial computational power.

8

Discussion

8.1 Test system

Unfortunately, there were a lot of issues and problems with the tests system which impeded the development of the controllers. The test vehicle was also used as a software checkout platform for another project at Delphi. These tests had higher priority than the thesis, which limited the ability to carry out testing of the controllers. If more time in the test vehicle would have been available, more testing could have been performed, which would ensure a high quality controller.

8.1.1 EPAS system

During the initial phase of the project, the interface to control the steering wheel angle over CAN was not fully understood. If the steering interface would have been functional at the start of the project, a lot more time could have been spent trying to deal with issues related to the control algorithms.

The existing EPAS system is only designated for LKA, where the demands on accuracy and resolution of the steering wheel angle are low. It was also observed that the EPAS has a dead zone close to zero degrees. When driving, the steering wheel is naturally forced back to zero degrees due to torque that comes from the wheels. It is when the servo motor changes from one rotational direction to another that the dead zone is present. The angular accuracy close to zero degrees is therefore low.

It would be desirable to fully measure and model the EPAS system with field tests, such that it can be included in the modeling and simulation.

8.1.2 Lane marker estimation

Unfortunately, a lot of time during real world testing was disturbed by poor performance of the test system. During the beginning of the project, the lane markers detection algorithm was not tuned, which led to the estimations being off. The estimations were inaccurate and contained a lot of noise. Once they were properly tuned, the control algorithms could be fully tested. This was however during the last week before the presentation, so there was not a lot of time available to fine tune the controllers. The gathered results were considered good enough to include in the

report and presentation. No scientific validation or tests were performed to actually ensure that the oscillations were in fact only due to the EPAS system. The lane marker estimations were however inspected visually. From this visual inspection, the estimations were concluded to be sufficiently accurate such that their noise and inaccuracy doesn't have any major impact on the controller performance.

8.1.3 Model parameters

The real vehicle tests and the simulations show that the mathematical model structure works for lateral control. But the testing also showed that the model parameters are not the same for the two cases and the tuning from the simulation did not work well in the real test. Despite this fact the parameters were not estimated for the test vehicle due to lack of time and access to a test track. Instead the controller were tuned to get a similar behavior in the test vehicle as in the simulation and the performance in the test vehicle would probably be better if the model parameters were estimated for the test vehicle.

8.2 Future work

One important thing to deal with if the project would proceed is to add something that takes the steering wheel dynamics into account. As stated before, the steering wheel controller is limited in its accuracy and precision, especially close to zero degrees. If this could be included in the modeling and control design, it might be possible to eliminate the oscillations.

Bibliography

[1] M. Abe, S. (.-b. collection), and B. (.-b. collection). *Vehicle Handling dynamics: theory and application*. English. 2nd edition. Amsterdam, [Netherlands]: Butterworth-Heinemann, 2015. ISBN: 0081003730.

[2] F. Borrelli et al. "MPC-based approach to active steering for autonomous vehicle systems". English. In: *International Journal of Vehicle Autonomous Systems* 3.2-3 (2005), pp. 265–291. ISSN: 1471-0226.

[3] *DARPA Urban Challenge*. URL: `http://archive.darpa.mil/grandchallenge/` (visited on 06/01/2016).

[4] A. Davies. *Baidu's Self-Driving Car Has Hit the Road*. 2015. URL: `http://www.wired.com/2015/12/baidus-self-driving-car-has-hit-the-road/` (visited on 06/01/2016).

[5] E. F. Drenth. *Brake Stability of Front Wheel Driven Cars at High Speed*. Tech. rep. 1993.

[6] J. F. Epperson. *An Introduction to Numerical Methods and Analysis, 2nd Edition*. English. 2nd ed. John Wiley & Sons, 2013. ISBN: 1118407466.

[7] T. Glad et al. *Control theory: multivariable and nonlinear methods*. English. London: Taylor & Francis, 2000. ISBN: 0203484754.

[8] *Google Self-Driving Car Project*. URL: `https://www.google.com/selfdrivingcar/faq/` (visited on 06/01/2016).

[9] B. Jacobson. *Vehicle Dynamics Compendium for Course MMF062*. Tech. rep. 267. 2015.

[10] M. Jalili-Kharaajoo. "Discrete-time sliding mode control design for linear time-varying systems: Application to steering control of highway vehicles". English. In: IEEE, 2004, pp. 113–116. ISBN: 9780780385993.

[11] R. N. Jazar, S. (service), and S. (.-b. collection). *Vehicle Dynamics: Theory and Application*. English. 2nd 2014.;2nd 2014; New York, NY: Springer New York, 2014. ISBN: 1461485436.

[12] M. LLC. *MS Windows NT Kernel Description*. 1999. URL: `http://web.archive.org/web/20080207010024/http://www.808multimedia.com/winnt/kernel.htm` (visited on 09/30/2010).

[13] B. McAleer. *Shifting Times: The rise of the automatic transmission*. Jan. 2015. URL: `http://driving.ca/chevrolet/corvette/auto-news/news/hail-hydra-matic-the-rise-of-the-automatic-transmission`.

[14] *mpcqpsolver - Solve a quadratic programming problem using the KWIK algorithm*. URL: http://se.mathworks.com/help/mpc/ref/mpcqpsolver.html (visited on 06/01/2016).

[15] J. Pauwelussen. *Essentials of vehicle dynamics*. English. 1st ed. Oxford, England: Butterworth-Heinemann, 2015. ISBN: 0081000367.

[16] *StateSpaceModel - Wolfram Language & Systems Documentation Center*. URL: https://reference.wolfram.com/language/ref/StateSpaceModel.html (visited on 06/05/2016).

[17] S.-S. You and S.-K. Jeong. "Automatic steering controllers for general lane-following manoeuvres of passenger cars using 2-DOF robust control synthesis". English. In: *Transactions of the Institute of Measurement and Control* 26.4 (2004), pp. 273–292. ISSN: 0142-3312.

A

Appendix 1

A.1 Matlab code for FHLQ controller

```
function iH = generateiH(A, B, Q)

% Inputs:
% A: Discrete transition matrix
% B: Discrete input matrix
% Q: Weight matrix
%
% Output:
% iH: Controller matrix

    H = zeros(N, N);

    for k = 1:N
        for i = 0:n-1
            for j = 0:n-1

                H(i+1, j+1) = H(i+1, j+1) + ...
                              B'*A^(n-1-i)'*Q*A^(n-1-j)*B;

            end
        end
    end

    iH = inv(H + H');

end
```

```
function u = FHLQ(Rc, Lc, vy, wz, Q, d, iH, stwhlAng)

% Inputs:
% Rc, Lc: Right and left lane coefficients
% vy: Lateral velocity
% wz: Yaw rate
% Q: Weight matrix
% d: Longitudinal distance points
% iH: Controller matrix
% stwhlAng: Current steering wheel angle
%
% Output:
% u: Latest control signal

    N = 20;     % Prediction horizon. This needs to be fixed at compile time

    Cc = (Lc + Rc)/2;   % The center coefficients

    theta = (Lc(3) + Rc(3))/2;  % Angle relative to lane at COG

    x0 = [-vy; wz; 0; -theta; stwhlAng];    % Current state

    R = zeros(5, N);    % Reference matrix

    fC = polyval(Cc, d);

    phi = Cc(3) + 2*Cc(2)*d + 3*Cc(1)*d.^2;

    R(3, :) = fC(2:N+1);
    R(4, :) = phi(2:N+1);

    f = zeros(N, 1);

    for n = 1:N
        for i = 0:(n-1)

             f(i+1) = f(i+1) + x0'*(A^n)'*Q*A^(n-1-i)*B ...
                             + (A^(n-1-i)*B)'*Q*A^n*x0 ...
                             - R(:, n)'*Q*A^(n-1-i)*B ...
                             - (A^(n-1-i)*B)'*Q*R(:, n);

        end

    end

    u = -iH(1, :)*f;    % Current control signal calculated
                        % by the first row of iH multiplied with f

end
```

A.2 Matlab code for Model Predictive Control

```
function [xv,y_r,phi_r,delta_r,td,status, uk] = MPC(C_coeffs,C_range,vx,...
    vy,wz,stwAng,stwRate,curv,c1,c0,dx,Nn,param, coeffs)
%#codegen

status = zeros(1,1,'double');

% Vehicle Parameters
% ------------------
m = param.m;              % Mass of vehicle
g = param.g;              % gravity
lf = param.lf;            % Length between CoG and front axle
lr = param.lr;            % Length between CoG and rear axle
L = lf + lr;              % Length between rear and front axle
J = param.J;              % Inertia around the z-axis
steer_ratio = param.steering_ratio; % ratio between steering wheel and
% front wheel


% Optimization & Modeling Coefficients
% ------------------------
w1 = coeffs.w1; % weigth on lateral displacement (py)
w2 = coeffs.w2; % weight on angular offset from reference (phi)
w3 = coeffs.w3; % weight on angular velocity (wz)
w4 = coeffs.w4; % weigth on lateral veolcity (vy)
w5 = coeffs.w5; % weigth on steering whell angle (delta)
w6 = coeffs.w6; % weight on fifth state

Nu = 4;  % Number of control steps (control horizon)
nx = 5;  % number of state variables
delta_rate = coeffs.delta_rate; % steering wheel ang. rate limit (rad/s)
delta_max = coeffs.delta_max;   % steering wheel angle limit (rad)
tFilt = coeffs.tFilt; % Filter time constant
tMPC = coeffs.tMPC;


% Calculate the approximate needed steering angle for feed forward
if abs(curv) > 0.00001
    Cf2 = c0*(m*g*lr/L) + c1*(m*g*lr/L)^2;
    Cr2 = c0*(m*g*lf/L) + c1*(m*g*lf/L)^2;

    R = 1/curv;

    Ku = (Cr2*lr-Cf2*lf)/(Cf2*Cr2*L);
    req_steer = L/R + Ku*m*vx^2/R;
else
   req_steer = 0;
end
```

```
% Reference Mapping in Space Domain
% --------------------------------

x_ref = linspace(dx,Nn,10);
N = length(x_ref); % prediction horizon in steps

y_ref = polyval(C_coeffs,x_ref); % Lateral samples space domain

xdiff = [ dx, diff(x_ref)]; % difference between each x-sample
ydiff = [ y_ref(1),  diff(y_ref)]; % differnce between each y-sample

% Approximative yaw angle at the samples
phi_ref = zeros(1,N);
for i=1:N-1
    phi_ref(i) = - atan2(ydiff(i+1),xdiff(i+1));
end
phi_ref(N) = phi_ref(N-1);

% C_der = [3*C_coeffs(1), 2*C_coeffs(2), C_coeffs(3)];
% tangent = polyval(C_der,x_ref);
% phi_ref = -atan(tangent);

tdiff = xdiff/vx;

if vx > 0

    % The cofficients of the C matrx have to be calculated
    C13 = (g*lf*lr*m*(4*c0^2*g*L^2*lr*(lf + lr) + ...
           c1*g*lr*m*(c1*g^2*lf*lr*(lf + lr)*m - 2*L^2*vx^2) + 2*c0*L*...
           (c1*g^2*lr*(lf + lr)^2*m - 2*L^2*vx^2)))/(4*J*L^4*tFilt*vx);

    C14 = (g*lr*(2*c0*L + c1*g*lr*m))/(2*L^2*tFilt);


    C23 = (g^2*lf*lr*(lf + lr)*m*(2*c0*L + c1*g*lf*m)*(2*c0*L + ...
          c1*g*lr*m))/(4*J*L^4*tFilt*vx);

    C24 = (g*lf*lr*m*(2*c0*L + c1*g*lr*m))/(2*J*L^2*tFilt);

    C31 = -((g^2*lf*lr*(lf + lr)* ...
          m*(2*c0*L + c1*g*lf*m)*(2*c0*L + c1*g*lr*m))/(4*J*L^4*tFilt));

    C32 = (g*lf*lr*m*(4*c0^2*g*L^2*lr*(lf + lr) + ...
          c1*g*lr*m*(c1*g^2*lf*lr*(lf + lr)*m - 4*L^2*vx^2) + 2*c0*L*...
          (c1*g^2*lr*(lf + lr)^2*m - 4*L^2*vx^2)))/(4*J*L^4*tFilt*vx);

    C33 = (g*lr*(2*c0*L + c1*g*lr*m))/(2*L^2*tFilt);

    C42 = (g^2*lf*lr*(lf + lr)*m*(2*c0*L + c1*g*lf*m)*...
        (2*c0*L + c1*g*lr*m))/(4*J*L^4*tFilt*vx);

    C43 = (g*lf*lr*m*(2*c0*L + c1*g*lr*m))/(2*J*L^2*tFilt);

    % The C matrix can then be expressed as:
    C = [0 0 C13 C14 0;
         0 0 C23 C24 0;
```

```
     C31 C32 C33 0 0;
     0 C42 C43 0 0];

%  C(3,:); % selction matrix for lateral displacement
H_py = kron(eye(N),2*w1*C(3,:)'*C(3,:));
f_py = -2*w1*kron(y_ref,C(3,:));

% C(4,:); % selection matrix for anlgular displacement
H_phi = kron(eye(N),2*w2*C(4,:)'*C(4,:));
f_phi = -2*w2*kron(phi_ref,C(4,:));

% C(2,:);  selection matrix for angular velocity
H_w = kron(eye(N), 2*w3*C(2,:)'*C(2,:));

% Cs_vy = C(1,:); % selection matrix for lateral velociy
H_vy = kron(eye(N),2*w4*C(1,:)'*C(1,:));

% Cs_5 = [0, 0, 0, 0, 1]; the fifth mathematica state
H_5 = kron(eye(N),2*w6*[0, 0, 0, 0, 1]'*[0, 0, 0, 0, 1]);


H_d = kron(eye(Nu),2*w5);
f_d = -2*w5*kron(ones(1,Nu),req_steer);


% H Matrix and f vector
% --------------

% min 1/2*x'*H*x + f*x

% min (ref_y(p+1) - t_y(p+1))^2, (ref_y(p+2) - t_y(p+2))^2
% min 2*w1*x'*Q_py*x - 2*w1*ysp*Cs_py*x

H = blkdiag((H_py + H_phi + H_w + H_vy + H_5), H_d);
f = [(f_py + f_phi), f_d];


% Equality Constraints
% --------------------

% Note: This A matrix is countinues time
A43 = -((g^2*lf*lr*m*(4*c0^2*L^2*(lf + lr)^2 + 2*c0*c1*g*L* ...
      (lf + lr)^3*m + c1*m*(c1*g^2*lf*lr*(lf + lr)^2*m + 2*L^2* ...
      (lf - lr)*vx^2)))/(4*J*L^4*tFilt*vx^2));

A44 = -(g*(4*c0^2*g*L^2*lf*lr*(lf + lr)^2*m*tFilt + 2*c0*L* ...
      (lf + lr)*(c1*g^2*lf*lr*(lf + lr)^2*m^2*tFilt + 2*L^2* ...
      (J + lf*lr*m)*vx) + c1*g*m*(c1*g^2*lf^2*lr^2*(lf + lr)^2* ...
      m^2*tFilt + 2*L^2*vx*(J*(lf^2 + lr^2) + lf*lr*m*(2*lf*lr + ...
      lf*tFilt*vx - lr*tFilt*vx)))))/(4*J*L^4*tFilt*vx^2);

A45 = -((2*c0*g*L*(lf + lr)*(J + lf*lr*m)*tFilt + c1*g^2*m*(J*(lf^2 ...
    + lr^2)+ 2*lf^2*lr^2*m)*tFilt + 2*J*L^2*vx)/(2*J*L^2*tFilt*vx));


A = [0 1 0 0 0;
```

```
     0 0 1 0 0;
     0 0 0 1 0;
     0 0 0 0 1;
     0 0 A43 A44 A45];


B = [0 0 0 0 1]';

% Calcukate map the current state to mathematica states
states = [vy wz 0 0]';
xk = C\states;

% System Matrices discrete time - Forward Euler
% (x(k+1) - x(k)/h = Ax(k) + Bu(k) => (I + hA)x(k) + hBu(k)
% ----------------------------

% Note 2: The system matrices are discretized
Ha = [zeros(nx,nx*N); eye(nx*(N-1)) + kron(diag(tdiff(2:end)), ...
    A), zeros(nx*N-nx,nx)];

Hi = eye(nx*N);
Hb = [kron(diag(tdiff(1:Nu)),B);...
    zeros(nx*(N-Nu),Nu-1), kron(tdiff(Nu+1:N)',B)];


Aeq = [Ha-Hi, Hb];
beq = [-(eye(nx)+tdiff(1)*A)*xk; zeros(N*nx-nx,1)];


% NOTE: USE THIS FOR mpcqpsolver => Ain*x >= bin

Ain = [kron([1; -1],[zeros(Nu,nx*N), eye(Nu) - ...
    [zeros(1,Nu); eye(Nu-1), zeros(Nu-1,1)]]); % 1) & 2)
    kron([1; -1], [zeros(Nu,nx*N), eye(Nu)])]; % 5) & 6)

bin = [(-delta_rate*tMPC+stwAng+stwRate*tFilt)/steer_ratio; ...
    -delta_rate/steer_ratio.*tdiff(1:Nu-1)'; % 1)
    (-delta_rate*tMPC-stwAng-stwRate*tFilt)/steer_ratio; ...
    -delta_rate/steer_ratio.*tdiff(1:Nu-1)'; % 2)
     -ones(2*Nu,1)*delta_max/steer_ratio]; % 5) & 6)

% 1) u(k+1) - u(k) >= - delta_rate*tdiff
% 2) u(k+1) - u(k) <= delta_rate*tdiff  (rad)
% 3) y(k) - y_ref(k) < lane_width/2 - car_width/2
% 4) y(k) - y_ref(k) > -(lane_width/2 - car_width/2)
% 5) u(k) > -delta_max
% 6) u(k) < delta_max


% Configuration mpcqpsolver
% -------------------------

opt = mpcqpsolverOptions('double');
[L,~] = chol(H,'lower');
Linv = inv(L);
iA0 = false(size(bin));
```

```
   [x,status] = mpcqpsolver(Linv,f',Ain,bin,Aeq,beq,iA0,opt);
    uk = x(nx*N+1)*steer_ratio;

else
    uk = 0;
    x = zeros(N*(nx+1),1);
end
xv = [x; zeros((nx+2)*N-length(x),1)];
y_r = [y_ref, zeros(1,N-length(y_ref))]';
phi_r = [phi_ref, zeros(1,N-length(phi_ref))]';
delta_r = req_steer;
td = [tdiff, zeros(1,N-length(tdiff))]';
end
```

B

Appendix 2

B.1 Simulation results

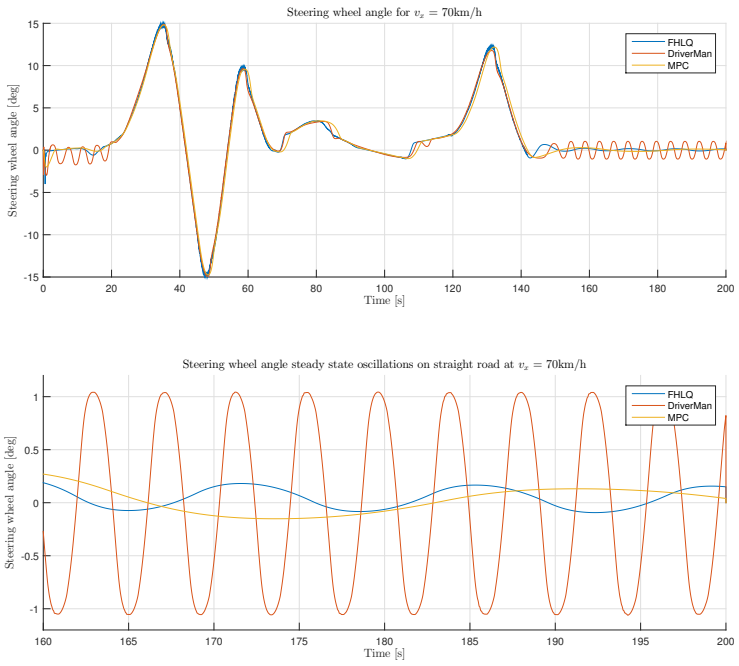


Figure B.1: This figure shows the requested steering wheel angle of the two controllers and CarMaker driver when driving on R_1 followed by a straight road. The longitudinal velocity is 70 km/h during the entire test.

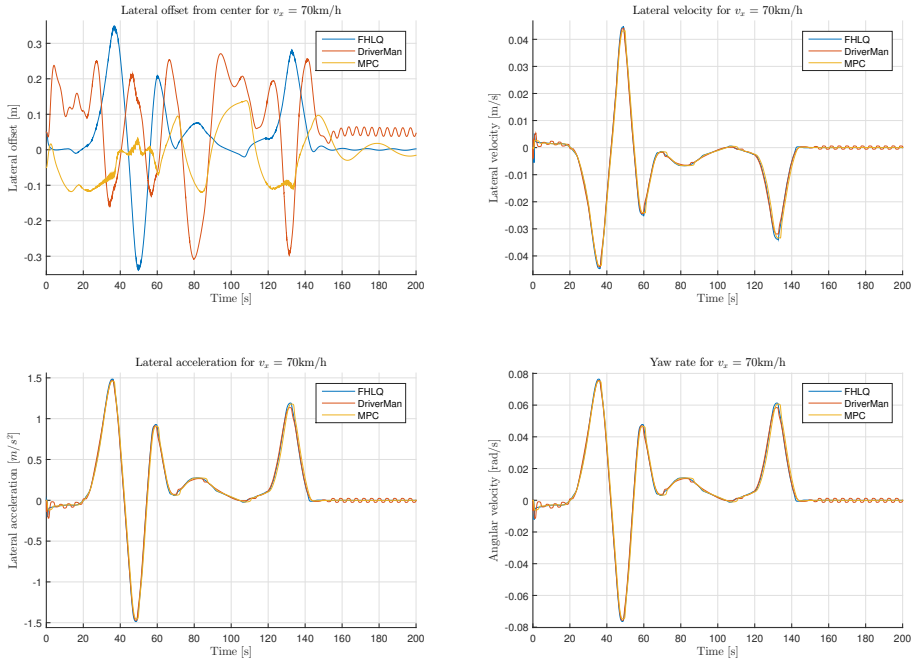


Figure B.2: This figure shows the controller performances in comparison to the CarMaker driver when driving on R_1 in 70 km/h.

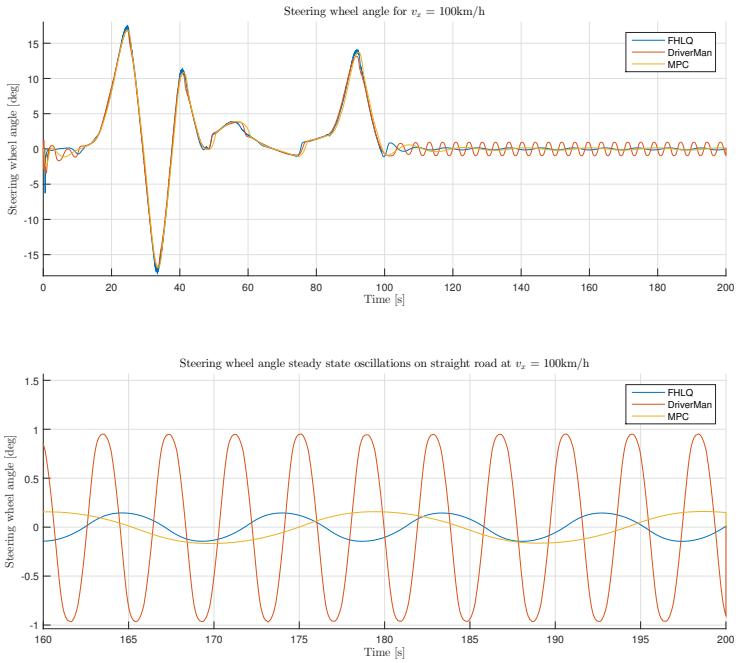


Figure B.3: This figure shows the requested steering wheel angle of the controllers and CarMaker driver when driving on R_1 followed by a straight road. The longitudinal velocity is 100 km/h during this test.

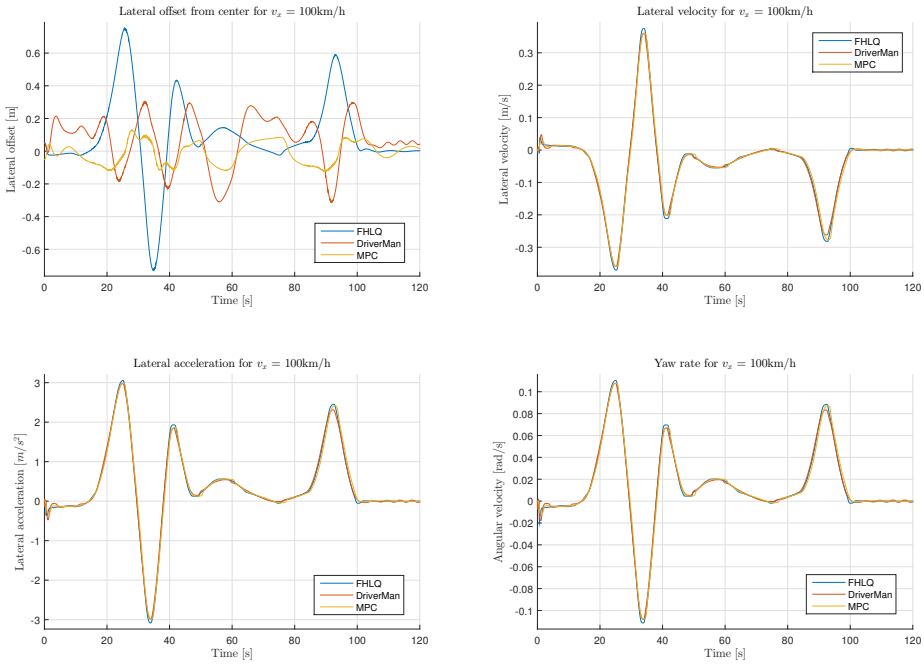


Figure B.4: This figure is connected to the previous and shows the performance of the controllers and CarMaker driver.

B.2 Model validation

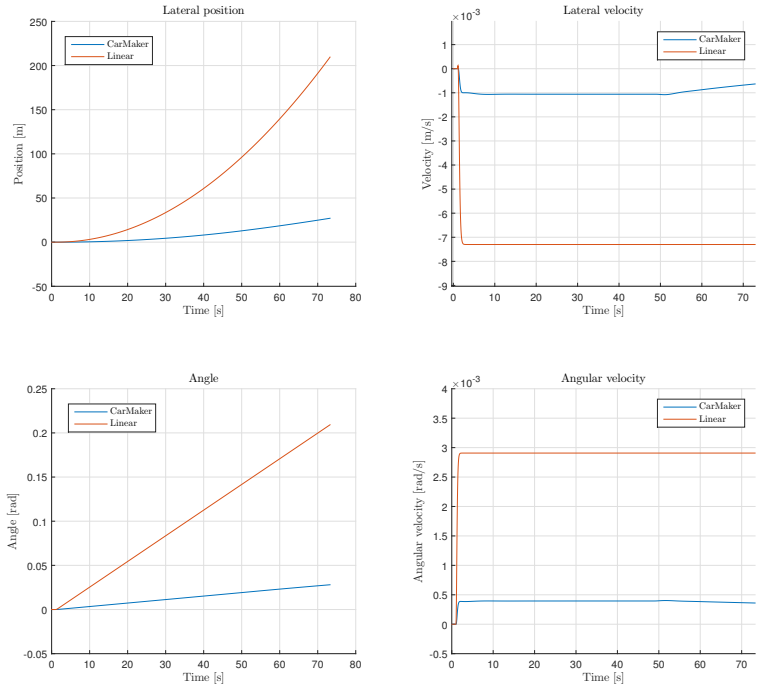


Figure B.5: In this set of figures is the steering angle request is a step signal with amplitude 0.01 rad. The linear model does not match the CarMaker model results very well.

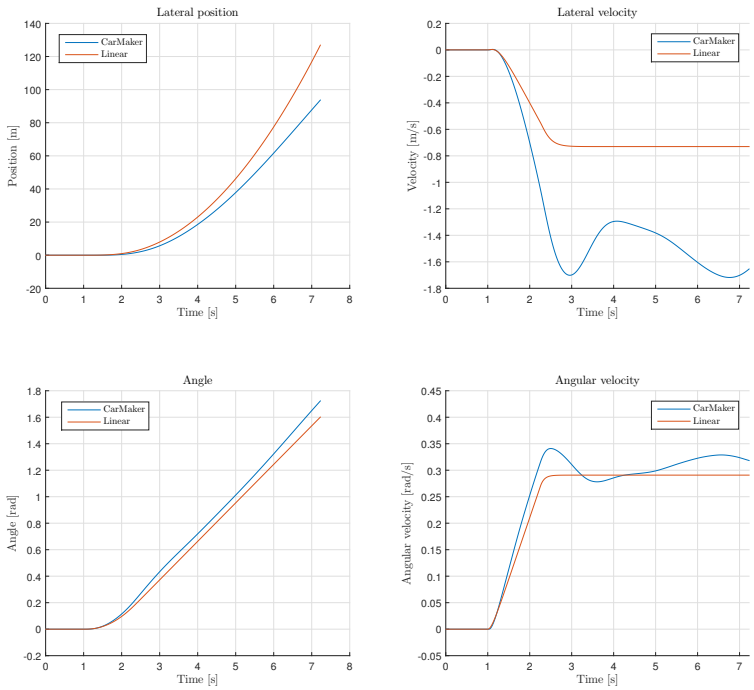


Figure B.6: In this set of figures are the steering wheel angle request a step signal with amplitude 1 rad. The nonlinear nature of the CarMaker model have a major impact on the lateral velocity and the angular velocity.

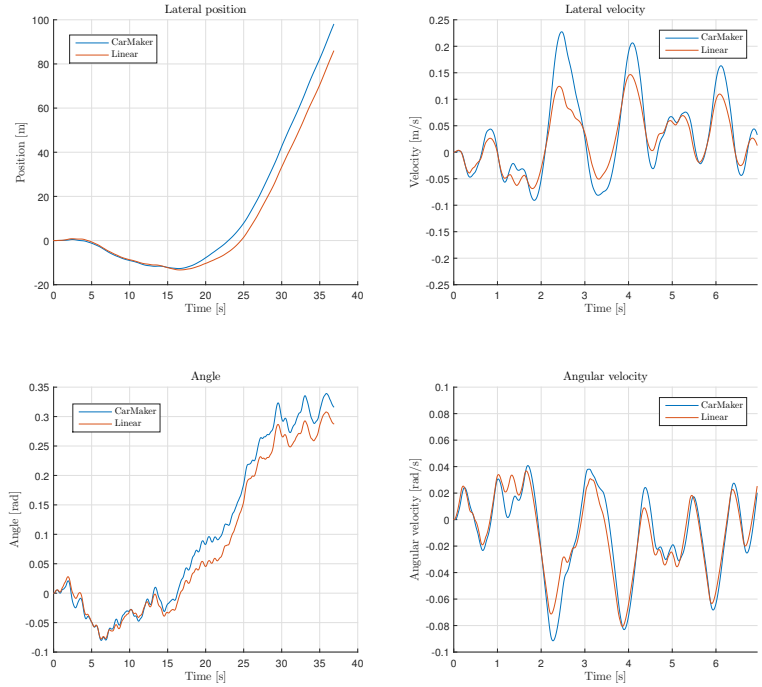


Figure B.7: In this set of figures is the steering wheel angle request random noise with a variance of 1 rad. As can be seen, the CarMaker model have greater peak values than the linear model.

C

Appendix 3

C.1 Mathematical model

If the state vector is formulated as

$$x = \begin{pmatrix} v_y \\ \omega_z \\ p_y \\ \psi_z \end{pmatrix}.$$

Then the continues time model matrix can be formulated as

$$A = \begin{pmatrix} -\frac{g(2c_0L(l_f+l_r)+c_1g(l_f^2+l_r^2)m)}{2L^2v_x} & \frac{c_1g^2l_f(l_f-l_r)l_rm-2L^2v_x^2}{2L^2v_x} & 0 & 0 \\ \frac{c_1g^2l_f(l_f-l_r)l_rm^2}{2JL^2v_x} & -\frac{gl_fl_rm(c_0L(l_f+l_r)+c_1gl_fl_rm)}{JL^2v_x} & 0 & 0 \\ 1 & 0 & 0 & -v_x \\ 0 & 1 & 0 & 0 \end{pmatrix}$$

and the corresponding input vector as

$$B = \begin{pmatrix} \frac{gl_r(2c_0L+c_1gl_rm)}{2L^2} \\ \frac{gl_fl_rm(2c_0L+c_1gl_rm)}{2JL^2} \\ 0 \\ 0 \end{pmatrix}$$

.

C.2 Mathematica generated model

With the following expressions

$$A_{43} = -((g^2 l_f l_r m(4c_0^2 L^2 (l_f + l_r)^2 + 2c_0 c_1 g L (l_f + l_r)^3 m + c_1 m(c_1 g^2 l_f l_r (l_f + l_r)^2 m + 2L^2 (l_f - l_r) v_x^2)))/(4JL^4 T v_x^2))$$

$$A_{44} = -((g(4c_0^2 g L^2 l_f l_r (l_f + l_r)^2 mT + 2c_0 L(l_f + l_r)(c_1 g^2 l_f l_r (l_f + l_r)^2 m^2 T + 2L^2 (J + l_f l_r m) v_x) + c_1 g m(c_1 g^2 l_f^2 l_r^2 (l_f + l_r)^2 m^2 T + 2L^2 v_x (J(l_f^2 + l_r^2) + l_f l_r m(2l_f l_r + l_f T v_x - l_r T v_x)))))/(4JL^4 T v_x^2))$$

$$A_{45} = -((2c_0 g L(l_f + l_r)(J + l_f l_r m)T + c_1 g^2 m(J(l_f^2 + l_r^2) + 2l_f^2 l_r^2 m)T + 2JL^2 v_x)/(2JL^2 T v_x))$$

the Mathematica generated model matrix can be expressed as

$$A_m = \begin{pmatrix} 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & A_{43} & A_{44} & A_{45} \end{pmatrix}$$

and the corresponding input vector is given as

$$B_m = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 1 \end{pmatrix}.$$

The coefficients of the Mathematica generated output matrix can be formulated as

$$C_{13} = (gl_f l_r m(4c_0^2 gL^2 l_r(l_f + l_r) + c_1 gl_r m(c_1 g^2 l_f l_r(l_f + l_r)m - 2L^2 v_x^2) + 2c_0 L(c_1 g^2 l_r(l_f + l_r)^2 m - 2L^2 v_x^2)))/(4JL^4 T v_x)$$

$$C_{14} = (gl_r(2c0L + c1gl_r m))/(2L^2 T)$$

$$C_{23} = (g^2 l_f l_r(l_f + l_r)m(2c_0 L + c_1 gl_f m)(2c_0 L + c_1 gl_r m))/(4JL^4 T v_x)$$

$$C_{24} = (glflrm(2c0L + c1glrm))/(2JL^2 T)$$

$$C_{31} = -((g^2 l_f l_r(l_f + l_r)m(2c_0 L + c_1 gl_f m)(2c_0 L + c_1 gl_r m))/(4JL^4 T))$$

$$C_{32} = (gl_f l_r m(4c_0^2 gL^2 l_r(l_f + l_r) + c_1 gl_r m(c_1 g^2 l_f l_r(l_f + l_r)m - 4L^2 v_x^2) + 2c_0 L(c_1 g^2 l_r(l_f + l_r)^2 m - 4L^2 v_x^2)))/(4JL^4 T v_x)$$

$$C_{33} = (gl_r(2c_0 L + c_1 gl_r m))/(2L^2 T)$$

$$C_{42} = (g^2 l_f l_r(l_f + l_r)m(2c_0 L + c_1 gl_f m)(2c_0 L + c1gl_r m))/(4JL^4 T v_x)$$

$$C_{43} = (gl_f l_r m(2c_0 L + c_1 gl_r m))/(2JL^2 T)$$

and the complete output matrix can be written as

$$C_m = \begin{pmatrix} 0 & 0 & C_{13} & C_{14} & 0 \\ 0 & 0 & C_{23} & C_{24} & 0 \\ C_{31} & C_{32} & C_{33} & 0 & 0 \\ 0 & C_{42} & C_{43} & 0 & 0 \end{pmatrix}.$$