

# Recursive feasibility of a hierarchical MPC structure for autonomous driving

Scientific thesis for the procurement of the degree B.Sc.  
from the Department of Electrical and Computer Engineering at the  
Technical University of Munich.

**Supervised by** Univ.-Prof. Dr.-Ing./Univ. Tokio habil. Martin Buss  
M.Sc. Tommaso Benciolini  
Chair of Automatic Control Engineering

**Submitted by** cand. ing. Ivan Nikolovski  
Soldauer Straße 26  
81927 Munich  
01781861183

**Submitted on** Munich, 29.10.2021



June 14, 2021

BACHELOR THESIS  
for  
Ivan Nikolovski  
Student ID 3717921, Degree EI

**Recursive feasibility of a hierarchical MPC structure for autonomous driving**

Problem description:

Model Predictive Control (MPC) is a leading approach for model-based control algorithms for autonomous driving. In [1] a hierarchical MPC structure was proposed, introducing an additional high-level layer acting as a maneuver planner. By using a simplified model and a long prediction horizon, this high-level controller optimizes over the choice of the reference trajectory for the low-level controller, allowing to reveal convenient maneuver choices that would prove to be efficient in the long run. However, recursive feasibility of the hierarchy was not addressed.

The recursive feasibility of the low-level controller can be ensured adopting standard approaches presented in literature [2], since the high-level layer influences the lower one only through the formulation of the reference trajectory. Conversely, as the high-level state is iteratively obtained by projecting the low-level one, the high-level problem feasibility cannot be handled separately. As a consequence, a proper choice of the constraints for the two levels is needed to guarantee the feasibility of the high-level optimal control problem. The aim of this work is to discuss the conditions on the constraints needed to ensure the feasibility of the whole control hierarchy, and to validate them through numerical simulations of an autonomous driving scenario.


Tasks:

- Literature research on recursive feasibility
- Development of high-level and low-level constraints to guarantee feasibility of the whole structure
- Discussion on the conditions guaranteeing feasibility of the control hierarchy, through extensive simulations in MATLAB

Bibliography:

- [1] T. Benciolini, T. Brüdigam, and M. Leibold. Multistage stochastic model predictive control for urban automated driving, 2021. *Submitted to the IEEE 24th International Conference on Intelligent Transportation Systems (ITSC)*.
- [2] A. Boccia, L. Grüne, and K. Worthmann. Stability and feasibility of state constrained mpc without stabilizing terminal constraints. *Systems & Control Letters*, 2014.

Supervisor: M. Sc. Tommaso Benciolini  
Start: 29.06.2021  
Intermediate Report: 03.08.2021  
Delivery: 07.09.2021

  
(M. Leibold)  
Akad. Oberrat





# Contents

<b>1</b>	<b>Introduction</b>	<b>5</b>
1.1	Related work . . . . .	5
<b>2</b>	<b>Technical approach</b>	<b>7</b>
2.1	Introduction to nonlinear model predictive control . . . . .	7
2.2	Basic idea of the 2 layer NMPC . . . . .	8
2.3	Vehicle models for the low-level layer . . . . .	8
2.3.1	Low-level EV model . . . . .	8
2.3.2	Low-level TV model . . . . .	9
2.3.3	Pedestrian model for the low level layer . . . . .	10
2.4	Vehicle models for the high-level layer . . . . .	11
2.4.1	High-level EV model . . . . .	11
2.4.2	High-level TV and pedestrian model . . . . .	11
2.5	Constraints for the low-level and high-level models . . . . .	12
2.5.1	Low-level constraints . . . . .	12
2.5.2	High-level constraints . . . . .	12
2.5.3	Determining the ellipse length and width . . . . .	13
2.6	Cost functions . . . . .	14
2.7	Recursive feasibility . . . . .	15
2.7.1	Constraints guaranteeing recursive feasibility . . . . .	16
<b>3</b>	<b>Simulation setup and results</b>	<b>19</b>
3.1	Simulation setup . . . . .	19
3.2	Simple simulation and setup . . . . .	19
3.3	Analysis . . . . .	21
3.4	Practical scenarios involving recursive feasibility . . . . .	22
3.4.1	Pedestrian scenario . . . . .	22
3.4.2	TV overtaking scenario . . . . .	23
3.5	Simulations including new constraints . . . . .	25
3.5.1	Pedestrian scenario . . . . .	26
3.5.2	TV overtaking scenario . . . . .	27
<b>4</b>	<b>Improvements and further development</b>	<b>31</b>

<b>List of Figures</b>	<b>33</b>
<b>List of Tables</b>	<b>35</b>
<b>Bibliography</b>	<b>37</b>

# Chapter 1

## Introduction

The main goal of this Bachelor's Thesis is to develop an algorithm for automated driving using nonlinear model predictive control with 2 layers of control. The first one being a low level layer which is necessary for the immediate decisions of the vehicle and the second one being a high level layer which plans the next steps for the low level layer as a maneuver planner. One could compare these 2 layers to a human, with the low level layer being our extremities and the high level layer being the brain. The second important part about this Bachelors Thesis is the recursive feasibility. Recursive feasibility is a term used in nonlinear model predictive control which means that we can guarantee not only that the current problem is solvable but also that the problems in the future will be solvable.

### 1.1 Related work

The most similar paper to the topic covered in the thesis was [BBL21]. In this paper it was described how to design a 2 layer model predictive control algorithm for urban situations. Similar to the thesis there were 2 layers in the algorithm with the same idea of one being the maneuver planner and the other the real time decision maker. My thesis also uses the same control structure of a 2 layer MPC algorithm. What makes the thesis different than this paper is the part about recursive feasibility which is to be implemented in the bachelor's thesis. Also in the thesis a more generalized high level algorithm was used which includes the decision for lane changing maneuvers.

Other works which were used in this bachelor's thesis include: a book on the basics of MPC [GP17], a paper on the topic of recursive feasibility [BGW14] and a paper on an autonomous driving algorithm using MPC with linear constraints [BOWL21].

There are also papers which proposed other solutions to the problems of autonomous driving. One such paper is "Motion planning for urban autonomous driving using Bezier curves and MPC" from X.Qian et al [QNdLFM16]. This paper also discusses lane changing maneuver using MPC. It also uses Bezier curves. Here an external



signal is given to the vehicle and it changes the lane whenever this signal is given based on the pre-calculated route by the Bezier curve. Another paper which delves into a similar problem is "Behavior and Path Planning Algorithm of Autonomous Vehicle A1 in Structured Environments" [KJK<sup>+</sup>13]. In this paper the problem of autonomous driving is again analyzed, but again a different solution is proposed. Here the algorithm is based on 3 different interconnected algorithms: a mission planner, path planner and longitudinal motion planner. The mission planner is responsible for respecting traffic regulations and keeping the lane. The path planner chooses the best path which the EV can take and the longitudinal planner decides when the vehicle has to accelerate or decelerate based on the situation. These 3 algorithms work intertwined with each other.

## Chapter 2

### Technical approach

#### 2.1 Introduction to nonlinear model predictive control

Nonlinear model predictive control (NMPC) is an optimization based method for the feedback control of nonlinear systems. Its primary applications are stabilization and tracking problems. Suppose we are given a controlled process whose state  $x(n)$  is measured at discrete time instants  $t * n$ ,  $n = 0, 1, 2, \dots$ . “Controlled” means that at each time instant we can select a control input  $u(n)$  which influences the future behavior of the state of the system. In tracking control, the task is to determine the control inputs  $u(n)$  such that  $x(n)$  follows a given reference  $x_{ref}(n)$  as good as possible. This means that if the current state is far away from the reference then we want to control the system towards the reference and if the current state is already close to the reference then we want to keep it there. The goal of the NMPC algorithm is to minimize a cost function  $J$  with a given stage cost  $l$ , subject to the state function of the system  $f$ . The key steps to the NMPC algorithm (as defined in [GP17]) are the following:

**Algorithm 1** (basic NMPC algorithm for constant reference  $x_{ref}$ )

1. Measure the state  $x(n)$  of the system.
2. Set  $x_0 = x(n)$  solve the optimal control problem (OCP)  
minimize

$$J_N(x_0, u(.)) = \sum_{k=0}^{N-1} l(x_u(k, x_0), u(k)) \quad (2.1)$$

with respect to  $u(.)$

subject to  $x_u(0, x_0) = x_0$ ,  $x_u(k+1, x_0) = f(x_u(k, x_0), u(k))$

and denote the optimal control sequence  $u^*(.)$ , with  $u(.)$  being a control sequence and  $x_u$  the state trajectory (state values from 0 to N).

3. Define the NMPC feedback  $\mu(x(n)) = u^*(.)$  and use the control value  $u(0)$  in the next sampling period.

The closed loop system resulting from the NMPC algorithm is defined as

$$x^+ = f(x, \mu_N(x)). \quad (2.2)$$

## 2.2 Basic idea of the 2 layer NMPC

The basic idea of the Bachelor's Thesis was to design a predictive algorithm for autonomous driving composed of 2 interconnected NMPC layers. The first layer also known as the low-level NMPC layer is based on the above described algorithm and is used for decisions regarding the low-level actions to guide the system in the short-term and to determine the immediate input which has to be applied to the system. The low-level has a small prediction horizon denoted by  $N$  and sampling time  $T$ . The second layer known as the high-level NMPC layer is also based on the above described algorithm and is used to decide the high-level actions ("maneuvers") to drive the system in the long term. The high-level layer has the same prediction horizon of  $N$ , but a higher sampling time of  $T_N$  which is calculated as

$$T_N = NT. \quad (2.3)$$

The higher sampling time is used to make sure that the ego vehicle predicts further in the future, which then allows the low-level NMPC algorithm to use the the first predicted step of the high-level algorithm as a reference for the low-level cost function. The high-level layer does not get updated to often since the high-level actions take a longer time to execute.

## 2.3 Vehicle models for the low-level layer

There will be 2 models used for the low-level layer (which were also used in [BOWL21]).

1. Model of the ego vehicle (EV): the kinematic bicycle model.
2. Model of the target vehicles (TV): the point mass model.

### 2.3.1 Low-level EV model

For the EV a discretized kinematic bicycle model was used. The following is the continuous kinematic bicycle model.

$$\dot{s} = v \cos(\phi + \alpha) \quad (2.4)$$

$$\dot{d} = v \sin(\phi + \alpha) \quad (2.5)$$

$$\dot{\phi} = \frac{v}{l_r} \sin(\alpha) \quad (2.6)$$

$$\dot{v} = a \quad (2.7)$$

$$\alpha = \arctan \left( \frac{l_r}{l_r + l_f} \tan(\delta) \right), \quad (2.8)$$

where  $s$  is the longitudinal position of the vehicle along the road,  $d$  is the lateral deviation from the center of the road (the center point of the middle lane),  $\phi$  is the yaw angle and  $v$  is the velocity,  $l_r$  and  $l_f$  are the distances from the vehicle center of gravity to the vehicle front and rear axis,  $\delta$  is the steering angle and  $a$  is the acceleration. The input vector is  $u = (a \ \delta)^T$  and the state vector is  $x = (s \ d \ \phi \ v)^T$ .

In order to speed up the computation we use a linearized model around the current state ( $x_0$ ) and the zero input (0). Moreover since the NMPC algorithm requires a linear discrete time model we obtain a discretized equivalent of the linearized model. As a result the predictions for the future states are obtained using the following model:

$$x_{k+1} = x_0 + T f^c(x_0, 0) + A_d(x_k - x_0) + B_d u_k, \quad (2.9)$$

with  $A_d$  and  $B_d$  being the discretized state and input matrices (see Appendix page 18 of [BOWL21]) and  $f^c$  being the continuous time model at state  $x_0$  and control input vector  $u$  equal to 0.

### 2.3.2 Low-level TV model

The actual state of the TV for the low level  $x_k^{TV}$  at time point  $k+1$  can be described as follows

$$x_{k+1}^{TV} = A x_k^{TV} + B u_k^{TV}, \quad (2.10)$$

with the actual input  $u_k^{TV}$  at time point  $k$  being

$$u_k^{TV} = \tilde{u}_k^{TV} + w_k^{TV}, \quad (2.11)$$

with  $\tilde{u}_k^{TV}$  being the controller for the control input and  $w_k^{TV}$  being the perturbation of the input. The perturbation of the input  $w_k^{TV}$  is a zero-mean Gaussian noise with covariance matrix  $\Sigma_w^{TV}$ . The reason why this noise is included is to propagate the uncertainty that may arise from the future movements of the other vehicles involved. The controller for the input  $\tilde{u}_k^{TV}$  can be calculated as follows

$$\tilde{u}_k^{TV} = K(\tilde{x}_k^{TV} - x_{ref,k}^{TV}), \quad (2.12)$$

with  $x_{ref,k}^{TV}$  being the TV reference. The predicted TV state for the low-level  $\tilde{x}_{k+1}^{TV}$  at time point  $k + 1$  is described as follows

$$\tilde{x}_{k+1}^{TV} = A\tilde{x}_k^{TV} + B\tilde{u}_k^{TV}. \quad (2.13)$$

$A, B$  and  $K$  are matrices with the following parameters

$$A = \begin{pmatrix} 1 & T & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & T \\ 0 & 0 & 0 & 1 \end{pmatrix} \quad (2.14)$$

$$B = \begin{pmatrix} 0.5T^2 & 0 \\ T & 0 \\ 0 & 0.5T^2 \\ 0 & T \end{pmatrix} \quad (2.15)$$

and

$$K = \begin{pmatrix} 0 & k_{12} & 0 & 0 \\ 0 & 0 & k_{21} & k_{22} \end{pmatrix}. \quad (2.16)$$

The state vector  $x$  consists of  $x = (x^{TV} \ v_x^{TV} \ y^{TV} \ v_y^{TV})^T$  and the input vector  $u$  consists of  $u = (a_x \ a_y)^T$ . We also consider a initial sensor noise in the first measurement of the TV state

$$\tilde{x}_0^{TV} = x_0^{TV} + w_0^{sens}, \quad (2.17)$$

which is also zero-mean with covariance  $\Sigma^{sens}$ .

### 2.3.3 Pedestrian model for the low level layer

A pedestrian model will also be used for both layers. The pedestrian will be designed in a similar way to the TV with some slight differences. The basic state space architecture will stay the same with the actual state of the pedestrian being

$$x_{k+1}^P = Ax_k^P + Bu_k^P, \quad (2.18)$$

with  $x_k^P$  being the state of the pedestrian and  $u_k^P$ . Due to the fact that the feedback matrix  $K$  (which was introduced previously for the low level model of the TV will) will be set to 0 for the pedestrian the actual input of the pedestrian will just be the noise

$$u_k^P = w_k^P, \quad (2.19)$$

which is denoted by  $w_k^P$ . Thus the predicted state of the pedestrian  $\tilde{x}_{k+1}^P$  will be

$$\tilde{x}_{k+1}^P = A\tilde{x}_k^P. \quad (2.20)$$

The matrices A and B stay the same as described in (2.14) and (2.15). The state vector  $x$  consists of  $x = (x^{TV} \ v_x^{TV} \ y^{TV} \ v_y^{TV})^T$  and the input vector  $u$  consists of  $u = (a_x \ a_y)^T$ . We also consider a initial sensor noise in the first measurement of the pedestrian state

$$\tilde{x}_0^P = x_0^P + w_0^{sens}, \quad (2.21)$$

which is zero-mean with covariance  $\Sigma^{sens}$ .

## 2.4 Vehicle models for the high-level layer

There are also 2 models used for the high-level layer. The TV model stays the same with the only difference being that the above given sampling time of  $T$  is exchanged for the high level sampling time of  $T_N$ . However the EV model used will be much simpler than the above mentioned bicycle model. This works since we do not require the most precise prediction for the maneuver planning. An approximate prediction suffices since we use the high level layer only for an estimate of where the EV should move within the next  $N$  steps of the simulation of the low-level layer. This estimate gets rounded up or down to the middle of the lane in which it is found, thus a precise estimate wouldn't result in a better simulation, it would just slow the simulation down and cause a longer computation time.

### 2.4.1 High-level EV model

The EV model for the high-level can be described using the following equations:

$$p_{k+1} = Gp_k + D\nu_k \quad (2.22)$$

with  $p$  being the LL state and  $\nu$  the low-level input and  $G$  and  $D$  being the identity matrices.  $p$  can be defined as  $p = (s \ d)^T$ , and  $\nu$  can be defined as  $\nu = (v \ \Delta)^T$ . With  $\Delta$  being the relative lane change to the previous time step.

### 2.4.2 High-level TV and pedestrian model

For the high level TV and pedestrian models we use the exact same low-level model. With the only difference being a different sampling time, i.e. we place  $T_N$  everywhere where  $T$  was used. Also to note this difference we add an extra HL notation in the inputs and states. For example we use  $x_{k+1}^{TV_{HL}}$  for the state in the  $k+1$ -th step instead of  $x_{k+1}^{TV}$  which was used for the low-level.

## 2.5 Constraints for the low-level and high-level models

### 2.5.1 Low-level constraints

The basic idea behind the constraints is to give the algorithm a set of rules which can never be violated. Thus they are mainly used for strict purposes, such as the vehicle always staying on the road or not exceeding a given speed.

The constraints used for the low-level model are the following. First of all there are the state constraints. The first one is a speed constraint of the low level which regulates the allowed lowest and highest speeds of the EV

$$v_{min} \leq v \leq v_{max}. \quad (2.23)$$

Then there is also a constraint regarding the lateral deviation so that the vehicle does not go off the road

$$d_{right} \leq d \leq d_{left}. \quad (2.24)$$

Then there are 2 constraints to both the inputs (i.e. the maximum and minimum allowed accelerations  $a$  and steering angles  $\delta$ )

$$u_{min} = [a_{min}, \delta_{min}] \quad (2.25)$$

$$u_{max} = [a_{max}, \delta_{max}]. \quad (2.26)$$

Then there is a ellipse-form constraint in immediate proximity of the TV which can be described using the following formula:

$$1 - \frac{s - x^{TV}}{l^2} - \frac{d - y^{TV}}{w^2} < 0, \quad (2.27)$$

with  $l$  and  $w$  being the wanted ellipse length and width, which will be derived in subsection 2.5.3. This ellipse form constraint is used to deter the EV from entering the TV immediate proximity to avoid crashes of the vehicles and also keep the safety of all vehicles involved.

### 2.5.2 High-level constraints

The high level constraints are similar to the low level ones with slight differences. Here there is a similar state constraint to the low-level model, regarding the lateral deviation

$$d_{right} \leq d \leq d_{left}. \quad (2.28)$$

This constraint as in the low-level model makes sure the EV does not go off the road. We have 2 input constraints also, one is for the speed  $v$  which is used as an

input in the high-level algorithm and the second one is for the relative lane change  $\Delta$

$$\nu_{min} = [v_{min}, \Delta_{min}] \quad (2.29)$$

$$\nu_{max} = [v_{max}, \Delta_{max}]. \quad (2.30)$$

The relative lane change  $\Delta$  constraint tells the simulation what is the maximum and minimum possible lane change in 1 high level simulation step. Similar to the low-level algorithm we also have the same ellipse form constraint for the TV, but with larger length and width

$$1 - \frac{s - x^{TV_{HL}}}{l_{HL}^2} - \frac{d - y^{TV_{HL}}}{w_{HL}^2} < 0. \quad (2.31)$$

This ellipse constraint serves a similar purpose to the low-level constraint, with one difference. This constraint is used in a way to plan a possible overtaking or lane swap in the next steps. This constraint is bigger than the low level one since the high-level layer predicts further in the future which results in more uncertainty in the position of the TVs.

### 2.5.3 Determining the ellipse length and width

The ellipse length and width for the low level algorithm can be determined using a stochastic approach, where we take the uncertainty of future movements of the TV into regard. We want to find a region around the predicted high level TV state which contains the true high level TV state with a certain probability  $\beta$ . The algorithm is no longer considered to be a NMPC problem rather a SMPC problem [BOWL21]. To derive the ellipse length and width stochastically we first need to find the error in the predicted high level TV state vs the actual high level TV state, which was mentioned above in subsection 2.3.2. From equations (2.10) and (2.13) we can derive the prediction error

$$e_k = \tilde{x}_{k+1}^{TV} - x_{k+1}^{TV}. \quad (2.32)$$

The TV prediction can now be written in terms of the prediction error as

$$\tilde{x}_{k+1}^{TV} = x_{k+1}^{TV} + (A + BK)e_k - Bw_k^{TV}, \quad (2.33)$$

which yields the prediction the error prediction update

$$e_{k+1} = (A + BK)e_k - Bw_k^{TV}. \quad (2.34)$$

Using (2.17) we can derive that the initial error is also a Gaussian 0 mean noise with covariance matrix  $\Sigma_0^e$ . Using this and the TV uncertainty  $\Sigma_w^{TV_{HL}}$  we can derive the rest of the error covariance matrices with the following recursive equation

$$\Sigma_{k+1}^e = B\Sigma_w^{TV}B^T + (A + BK)\Sigma_k^e(A + BK)^T. \quad (2.35)$$



Since for the ellipse semi-major and semi-minor axes we only require the position of the TV we take a reduced error and covariance matrix containing only the positions of the TV  $\tilde{e} = (e_{x,k} \ e_{y,k})^T$  and  $\Sigma_{k+1}^{\tilde{e}} = \begin{pmatrix} \sigma_{x,k} & 0 \\ 0 & \sigma_{y,k} \end{pmatrix}$ . From here to calculate the ellipse semi major and minor axis we only need to include the previously mentioned safety factor  $\beta$  in the equation

$$\kappa = -2\ln(1 - \beta). \quad (2.36)$$

This ensures that TV state lies within our ellipse with a probability  $100\%\beta$ . Using this equation we can adjust the size of the ellipse dependent on how sure we want to be that the TV is contained inside it. As for the final semi minor and major axes of the ellipse we can define them as following

$$e_{x,k,\kappa} = e_{x,k}\sqrt{\kappa} \quad (2.37)$$

$$e_{y,k,\kappa} = e_{y,k}\sqrt{\kappa}. \quad (2.38)$$

One last thing to note is that using the exact same procedure we can derive the ellipse length and width for the high-level algorithm.

## 2.6 Cost functions

The main goal of the cost function is to give the algorithm a function which it tries to minimize, thus we can adjust the cost function to our needs and prioritize what we want the algorithm to do. The cost function of the low-level algorithm is completely dependent on the optimal control sequence of the high level algorithm. That is why it is important to first explain the cost function of the high-level algorithm. The high-level algorithm has the following cost function

$$J_{HL} = \sum_{k=0}^{N-1} (p - p_{ref})^T Q_{HL} (p - p_{ref}) + (\nu - \nu_{ref})^T R_{HL} (\nu - \nu_{ref}). \quad (2.39)$$

The function of the reference  $p_{ref}$  and  $\nu_{ref}$  is to allow the user to set reference values for the state and input which the algorithm will try to uphold. This forces the algorithm to follow the instructions given from the references, since the algorithm tries to minimize the cost function under consideration of the constraints (the constraints can not be violated).

After the high-level algorithm simulation is executed, the low-level algorithm takes the first input of the optimal control sequence  $\nu^*(0)$  and state trajectory  $p_e(1, p_0)$  and uses them as a reference for the low level cost function of the state. The important thing to note here is that we only use the speed and lateral deviation of the high-level algorithm and not the relative lane change  $\Delta$ , since the relative lane

change is not measured in the low-level algorithm and we do not want to penalize the relative lane change in the cost function. So using all of the given information we can summarize the reference  $x_{ref}$  for the cost function of the low-level as follows

$$x_{ref} = \begin{pmatrix} 0 & p_\nu(0, p_0)(2) & 0 & \nu^*(0)(1) \end{pmatrix}^T, \quad (2.40)$$

with  $p_e(1, p_0)(2)$  symbolizing the second element of the first predicted state trajectory of the high-level i.e. the lateral deviation  $d$  and  $\nu^*(0)(1)$  being the first element of the first predicted optimal control sequence i.e. the predicted speed  $v$ . Having derived all of the necessary values for the low-level, we can now implement the low-level cost function as follows

$$J = \sum_{k=0}^{N-1} (x - x_{ref})^T Q (x - x_{ref}) + (u - u_{ref})^T R (u - u_{ref}). \quad (2.41)$$

Important to note is that the reference  $x_{ref}$  as a variable is completely dependent on the result of the optimal control sequence and state trajectory of the high-level algorithm. This is how the two layers are interconnected with each other.

The matrices  $Q$ ,  $R$ ,  $Q_{HL}$  and  $R_{HL}$  are known as weighting matrices. The weighting matrices are always diagonal matrices. The choice of their values is not limited except that the first diagonal entry of the matrices  $Q_{HL}$  and  $Q$  is 0, since we do not want to penalize the algorithm in any way for the longitudinal deviation, so we always set  $Q(1, 1) = 0$  and  $Q_{HL}(1, 1) = 0$ . All the other choices for the weighting matrices are free since we can tell the algorithm in that way what we want it to prioritize, whether it is speed lateral deviation or yaw angle.

## 2.7 Recursive feasibility

The optimal control problem is called feasible if the set over which we optimize for a initial state  $x_0$  is nonempty. In other words if there is a solution possible for the given initial state. Recursive feasibility means that if a given state  $x$  is feasible then we can guarantee that it's closed loop successor  $f(x, \mu_n(x))$  is again feasible. The problem with recursive feasibility is not necessarily the implementation, but rather finding the constraints which can ensure that the next states are feasible. Especially in our case where we have 2 optimal control problems which have to both be feasible for the algorithm to work. For the low level layer feasibility can be relatively easily be ensured if the right constraints are applied, because the low level is only influenced by the high level due to the reference given by the cost function. The main difficulty is making sure that the high level is always feasible. Infeasibilities can occur easily in the high level layer due to the mismatch between predicted position and actual position since the high level layer gets updated only once in  $N$  iterations and the state of the EV is dependent on the low level. Thus the predictions don't always match the actual result in the end, due to many factors such

time required to accelerate, decelerate or changes of the low level state of the TV or pedestrian. This mismatch in prediction and end result might cause an infeasibility. This can of course be avoided if the right constraints are implemented to the low level and they not only ensure low level feasibility, but also high level feasibility.

### 2.7.1 Constraints guaranteeing recursive feasibility

In order to ensure that recursive feasibility can be guaranteed constraints need to be developed. These constraints need to make sure that assuming a initial state which does not violate the constraints the next state will also be feasible. There are multiple ways to ensure this does not happen. In the algorithm linear constraints will be used. The reason why linear constraints are chosen is quite simple. The easiest way to ensure feasibility is to make sure that the EV does not enter the safety ellipse of the TV. Thus the next step would be deriving a stopping distance which is large enough to make sure the EV stops before entering the safety ellipse and expanding it for the next prediction steps. In the case of the EV and TV being side by side (in lanes which are neighboring) we will need a horizontal constraint, which keeps the EV a certain safe distance away from the safety ellipse. In the next part of this section it will be shown how we can derive these constraints.

The vertical constraint can be derived from the equations for linear accelerated motion:

$$v = l + at \quad (2.42)$$

and

$$s = lt + \frac{at^2}{2} \quad (2.43)$$

,with  $l$  being the initial speed and  $a$  being acceleration. If we set the speed  $v = 0m/s$  in equation (2.42) we can derive the time needed to stop which we can then plug in the second equation (2.43) to find out the stopping distance. However we also need a constraint for the other steps in the prediction horizon. For this we assume maximum deceleration at each step and shift the constraint forward using the following equation:

$$v = (l - aTr) + at \quad (2.44)$$

Setting  $v = 0m/s$  gives us the wanted time to come to a stop

$$t_{stop} = \frac{aTr - l}{a} \quad (2.45)$$

,with  $r$  being the number of iterations and  $T$  being the sampling time between steps. The only thing left to do is plug in the time  $t_{stop}$  in 2.43 which then gives us the stopping distance  $x_{stop}$

$$x_{stop} = \frac{-l^2}{2a} + \frac{aT^2r^2}{2} \quad (2.46)$$

However to this constraint we will add an extra term. This term will be the semi-major ellipse length derived in the previous sections.

$$x_{vert} = x_{stop} + e_{x,k,\kappa} \quad (2.47)$$

For the horizontal constraint we will add an arbitrary safe distance to the length of the safety ellipse.

$$y_{hor} = y_{safe} + e_{y,k,\kappa} \quad (2.48)$$

These 2 constraints will be applied in specific situations. This can be done using multiple **if** clauses. They will be given to the constraint function of the solver as one constraint

$$q_x x + q_y y + q_t < 0. \quad (2.49)$$

Dependent on the situation the constraint will be either vertical or horizontal. The vertical constraint will be applied if there is a TV in the same lane as the EV in immediate vicinity of the EV ( $< 30m$ ) or if there is a pedestrian with a  $x$  position larger than the TV's  $x$  position and a  $y$  position which is  $1m$  away from the edge of the road. The horizontal constraint will be applied in a situation where there is a TV 1 lane away from the EV and has a  $x$  position which is larger than the EV's  $x$  position.

We also want to make sure that the high level maneuver planner always has a feasible state in the next step and thus ensure recursive feasibility for the high level. A simple way to ensure this is to make sure that there is always enough distance for the EV to stop in case of it coming to close proximity to a TV, very much like the low level. This again can be done in a very similar fashion to the above derived linear constraint for the low level with substituting  $T$  for  $T_N$  and adjusting the maximum allowed deceleration for the high level to be

$$a_{HL} = \frac{T_N a}{T}. \quad (2.50)$$

Setting this in the previously derived equation we can get the stopping time for the high level

$$x_{stopHL} = \frac{-l^2}{2a_{HL}} + \frac{a_{HL} T_N^2 r^2}{2} \quad (2.51)$$

However in practice we do not need a extra constraint to ensure this, since with the above mentioned constraints for the low level we already make sure that the EV has enough place to stop in case of a sudden braking maneuver by a TV. Since the

length of the prediction horizon for the low level is  $N$  and 1 high level iteration is the length of  $T_N = NT$  and the above mentioned constraints (2.47),(2.48) always hold true for  $N$  steps, thus recursive feasibility is always guaranteed for the next high level step.

## Chapter 3

# Simulation setup and results

### 3.1 Simulation setup

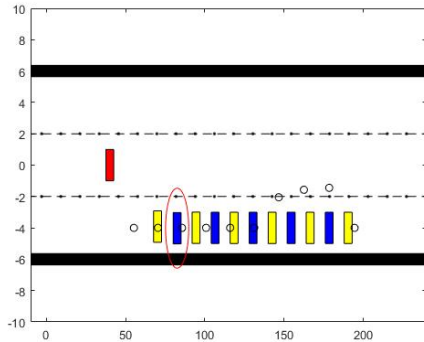
For the simulation setup a simple highway scenario will be used with three lanes. Each lane has the width 4 and a theoretical infinite length. In the simulations there will be 1 TV and 1 EV. The EV will be moving at a higher speed than the TV since this will allow us to see how the algorithm is working. The safety factor  $\beta$  will be set to  $\beta = 0.6$ . The values used for the state constraints of the low-level are  $v = [0m/s, 13m/s]$ ,  $d = [-6m, 6m]$ . The values of  $6m$  and  $-6m$  for the lateral deviation are derived from the furthest points on the road. For the inputs constraint of the low-level the values used will be  $a = [-9m/s^2, 5m/s^2]$  and  $\delta = [-0.4rad, 0.4rad]$ . For the high level we will set the lateral deviation constraints to  $d = [-4.25m, 4.25m]$ . The input constraints will be set to  $v = [0m/s, 13m/s]$  and  $\Delta = [-4m, 4m]$ . For the simulations setup we will set  $\nu_{ref} = (8m/s \ 0m)^T$  and  $p_{ref} = (0m \ -4m)^T$ .

### 3.2 Simple simulation and setup

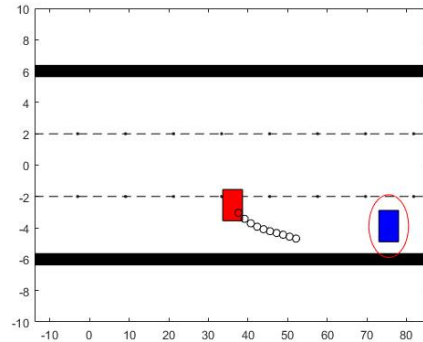
In this part of the thesis we will run a simple simulation to show how the algorithm works. There will be figures provided of both levels. Important to note here is that the low-level algorithm layer is what is happening in real-time whereas the high-level is the algorithms "brain" which gives the low-level layer the instructions on how to proceed further. The EV starting position will have the following state coordinates  $x = (40m \ 0m \ 0rad \ 8m/s)^T$ , whereas the TV will have these coordinates  $x^{TV} = (70m \ 6m/s \ -4m \ 0m/s)^T$ .

In the figures below we will see two vehicles, the EV and TV. The EV will be colored red whereas the TV will be colored blue. We also will see both the low level and high level simulations. They look the same with the only difference being that the high level simulation predicts further in the future and thus we will see a larger distance and in the higher level there will also be a the predicted future position of the TV,

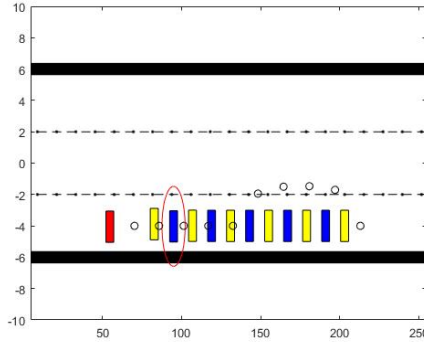
which for clarity will be marked with alternating blue and yellow colors. The red ellipses in both the simulations are the safety ellipses. In the high level simulation the ellipse will be on the second vehicle, since we only use the high level simulation for the second predicted position of the EV (which is used as a reference for the low level), for which we need the corresponding predicted position of the TV. There will be white dots after the EV in both simulations which represent the predicted future positions of the EV.



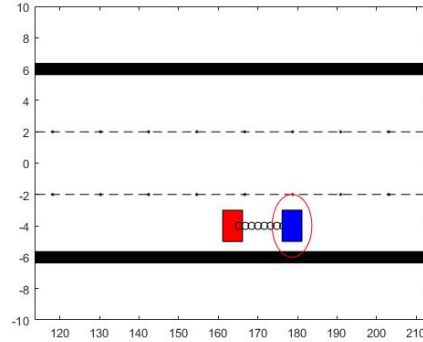
(a) Lane switching (High level)



(b) Lane switching (Low level)



(c) Next planned maneuvers (High level)



(d) Center of the lane (Low level)

Figure 3.1: Following the reference

The first thing the EV does is switch to the right lane. This happens since we set the high level lateral reference to  $-4m$  which corresponds to the center of the right lane. This is also shown in figure 3.1(a) as the planned maneuver is to force the EV to go to the right lane. This is successfully achieved as we can see in figure 3.1(b). In figure 3.1(d) we can see that after the lane switch the EV successfully stays in the middle point of the right lane. In figure 3.1(c) we can see the future predictions of the movement of the EV and TV.

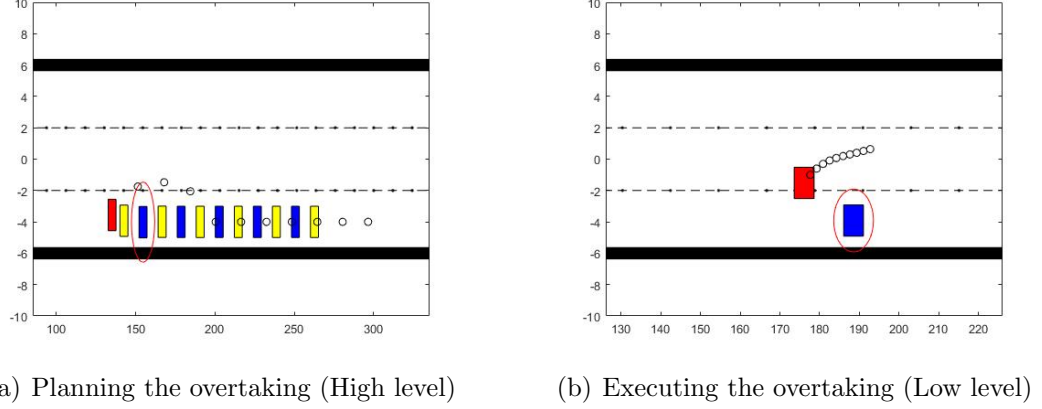


Figure 3.2: Overtaking

The next logical thing to do is to overtake the TV in front of the EV, since the EV is moving at a higher speed than the TV. As we can see in the high-level simulation figure 3.2(a) there is a maneuver planned at the next step for the low-level to again switch back to the middle lane. In figure 3.2(b) it is shown that this maneuver is successfully executed by the low level.

The last step is the EV has to come back to the same lane it started at since this minimizes the cost function. For this to happen the high-level layer plans this step in advance as we can see in the figure 3.3(a) and then the low-level layer executes it as we can see in the figures 3.3(b) and 3.3(c). In a realistic driving scenario this step might not always make sense since we do not always want to go back to the same lane. In the algorithm it makes sense since we want to minimize the cost function and we set the right lane as our reference.

### 3.3 Analysis

From the figures shown above we can see that the simulation works and is successfully able to do the complete the desired maneuver as well as keep the lane specified by the cost functions unless it is forced to switch the lanes by the constraints. This means that the wanted goal has been achieved and the simulation works successfully. Another thing to note is that it might seem like the low level algorithm doesn't exactly execute the maneuvers planned by the high level layer. This is somewhat true, since in the code there is an **if** clause included which forces the EV to go to the center of the lane for the reference calculated by the high level algorithm.



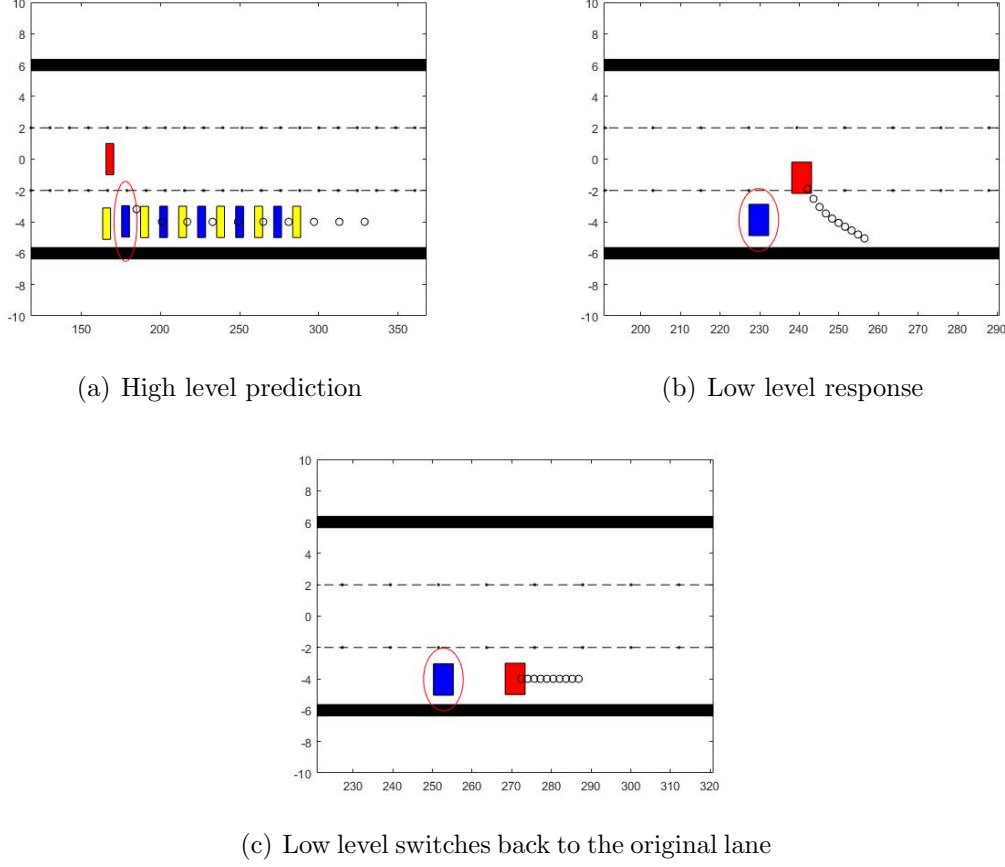


Figure 3.3: Switching back to the original lane

## 3.4 Practical scenarios involving recursive feasibility

In this section we will take a look at a couple of practical problems which are caused if recursive feasibility is not ensured.

### 3.4.1 Pedestrian scenario

For the first simulation we will take a look at a simple scenario of a pedestrian crossing the road and an EV which initially thinks it can cross the road and then suddenly changes its "mind" seeing that it can no longer cross before the pedestrian. For the simulation we will have a pedestrian crossing the road at a speed of  $1m/s$  and EV moving forward at an initial speed of  $3.3m/s$ .

As we can see here in figure 3.4 the high level planner thinks it can pass before the pedestrian crosses the road. The low level also tries to execute this as seen in figure

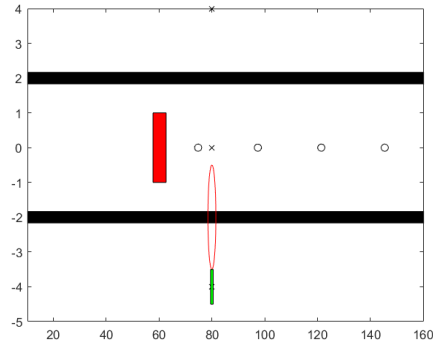


Figure 3.4: Initial position (High level)

3.5 and avoid the pedestrians safety ellipse.

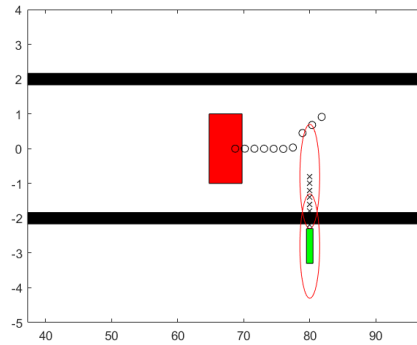


Figure 3.5: Position after accelerating (Low level)

However in practice the high level planner thinks the EV can immediately reach the proposed speed, which of course isn't the case since we need some time to accelerate to the proposed speed by the high level planner. Due to this the EV actually finds itself in a position which is a bit behind than the planner anticipated as we can see in figure 3.6.

This of course causes a problem since now the EV can't stop in time to not hit the pedestrian, but also can't accelerate before the pedestrian which causes an error in the high level planner and the constraints are violated.

### 3.4.2 TV overtaking scenario

Another example of a problem caused by feasibility is the following scenario. Just to ensure that the predictions of the TVs do not overlap with one another a prediction

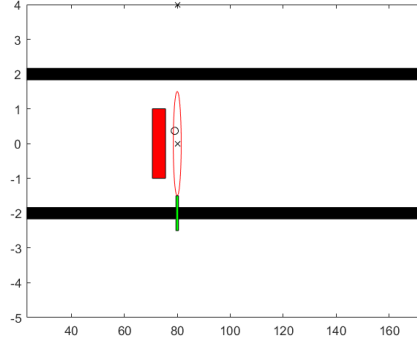
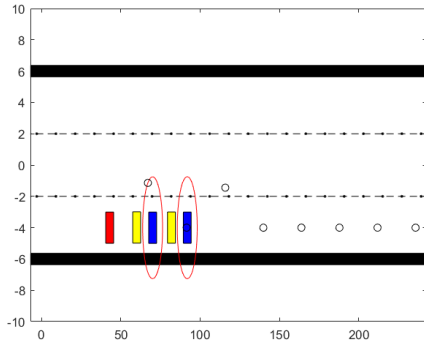
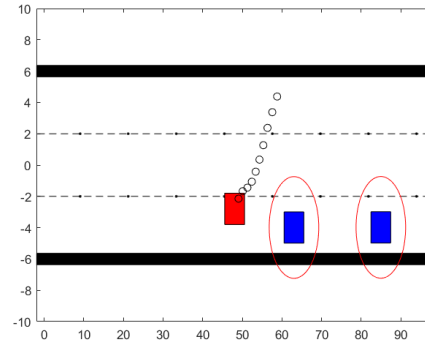


Figure 3.6: Position after accelerating (High level)

of only the first predicted position will be shown and it's safety ellipse which will be colored in blue, whereas the current position of each TV will be yellow. We will have 2 TVs moving at a speed of  $5m/s$  in the right lane and a EV in the same lane moving at a speed of  $12m/s$  as we can see in figures 3.7(a) and 3.7(b).



(a) Starting position (High level)

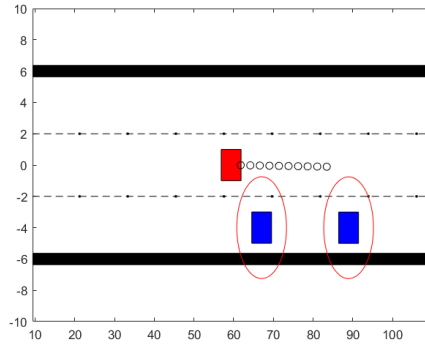


(b) Starting position (Low level)

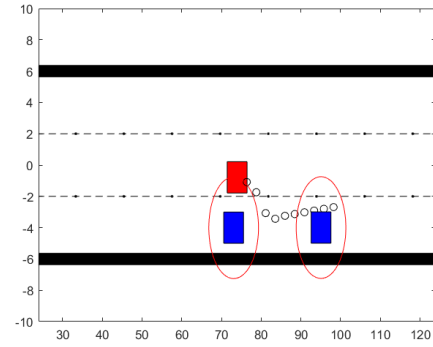
Figure 3.7: Starting positions of the high and low level

As we can see in figure 3.7(a) the EV plans to first overtake the first TV and switch to the middle lane.

This maneuver is successfully executed and no infeasibilities occur yet. This is shown in figure 3.8(a). The infeasibility occurs in the next couple of steps when the EV decides it is time to switch back to the original lane as seen in figure 3.9(a), but the TV decides it also wants to switch to the middle, the high level planner doesn't anticipate this and thus the EV continues to the right lane, whereas the TV tries to go to the middle lane, which results in the EV being inside the safety ellipses of both the TVs as depicted in figure 3.9(b). This of causes an infeasibility error.

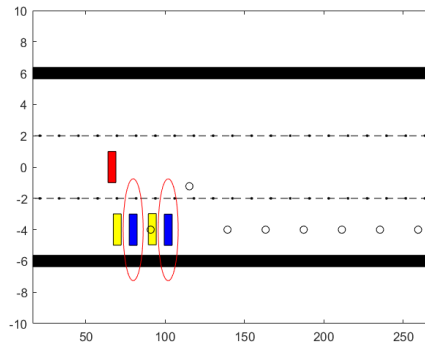


(a) EV goes to the middle lane (Low level)

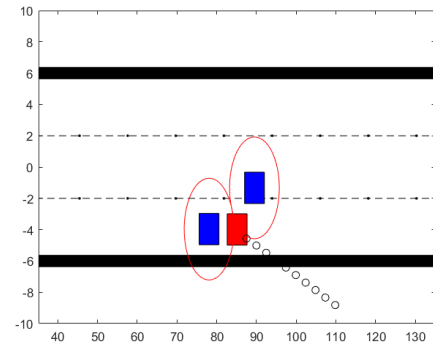


(b) EV tries to go back to the original lane (Low level)

Figure 3.8: Overtaking maneuver



(a) Solution (High level)



(b) Unexpected maneuver from the TV (Low level)

Figure 3.9: Occurrence of the error

### 3.5 Simulations including new constraints

In this section we will take a look at how the same situation which were described above are changed if we implement linear constraints which guarantee recursive feasibility.

### 3.5.1 Pedestrian scenario

Regarding the pedestrian scenario the linear constraints will make sure that EV keeps enough distance if the pedestrian is in the vicinity of the road. This of course makes the system more conservative but ensures that there won't be any crashes or casualties.

If we take the same starting situation as above with the only difference being the linear constraints to the lower level, we will get an outcome which doesn't result in an error due to linear constraint applied to the low level. This can be seen in the following figures. The simulation starts out the same with the high level proposing an acceleration as seen in figure 3.10. However this time due to the linear constraint the EV only accelerates to the point until the linear constraint gets applied and then starts slowing down, due to the linear constraint. The linear constraint and slowing down can be seen in figure 3.11

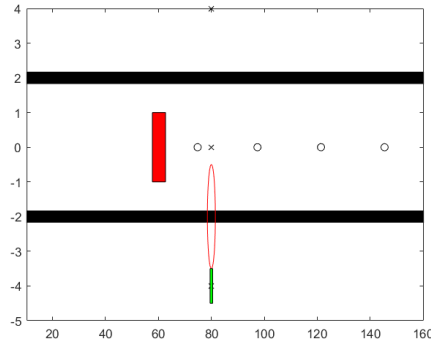


Figure 3.10: Starting position (High level)

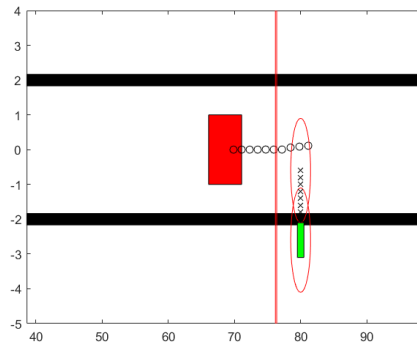


Figure 3.11: Constraint applied (Low level)

As stated above the EV will slow down due to the linear constraint, but it won't end up causing an error for the high level. We can see that in figure 3.12 and also

compare to how this differs to the error caused by the high level seen in figure 3.6.

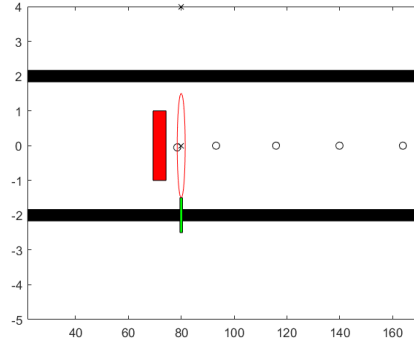


Figure 3.12: Position after accelerating with low level constraints (High level)

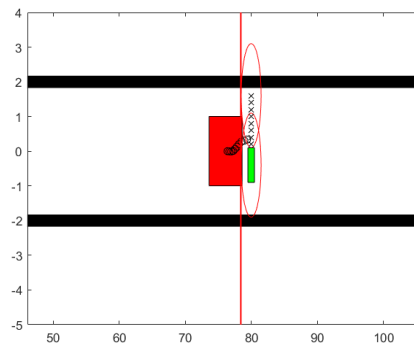


Figure 3.13: EV comes to a stop (Low level)

Eventually the EV will also stop due to the linear constraint and the pedestrian in the immediate vicinity, as seen in figure 3.13. This also increases the cost for the cost function, but safety is always ensured which is the most important goal for us.

Once the pedestrian crosses the road the linear constraint will be lifted and the TV can cross afterwards. This can be seen in figure 3.14.

### 3.5.2 TV overtaking scenario

If we take a look at the same situation which was described above with the 2 TVs in front of the EV and apply the same linear constraints which we derived in the previous chapter. We will again get a feasible result.

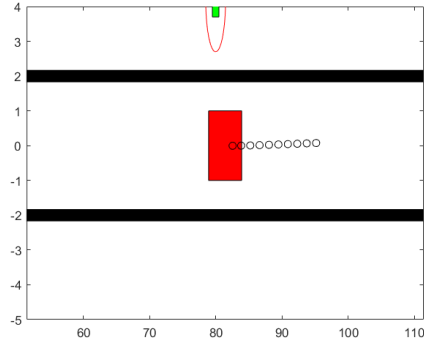
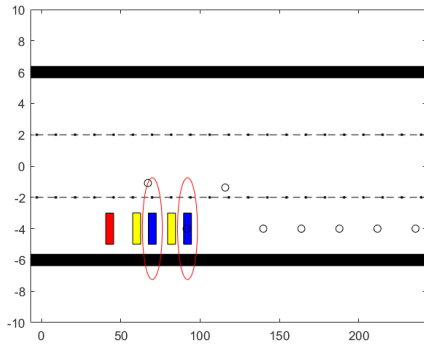
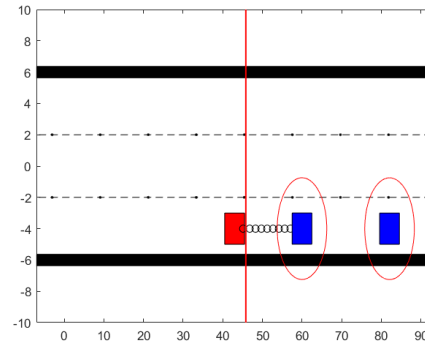


Figure 3.14: EV moves forward (Low level)

As we can see in figure 3.15(a) the starting proposal from the high level is the same, it again suggests overtaking by switching to the middle lane.



(a) Starting position (High level)

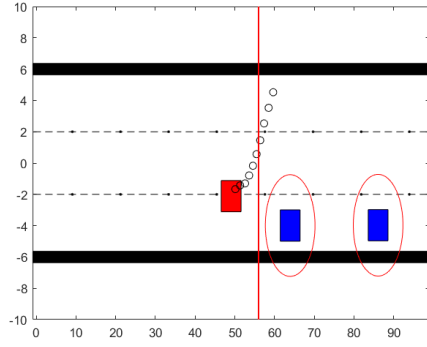


(b) Starting position (Low level)

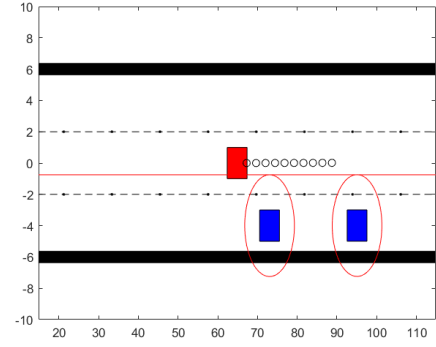
Figure 3.15: Starting positions

This is then successfully executed (as seen in figure 3.16(a)) and the vertical constraint switches now to a horizontal constraint as shown in figure 3.16(b). This is the main reason why this simulation will end up resulting in a feasible solution at the end, because this deters constraint a preemptive switch back to the lowest lane.

The next events which occur are again the second TV switching to the middle lane and the constraint switches from a horizontal to a vertical constraint whereas the other horizontal constraint stays there due to the other TV. The EV then stays in place due both the constraints given by the low level resulting in a feasible end result.

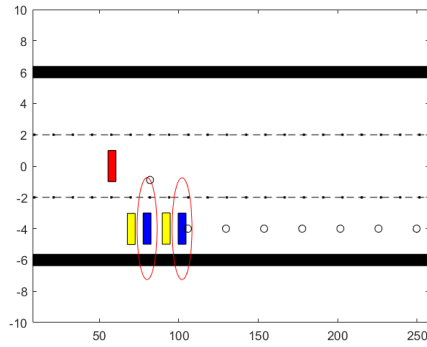


(a) Overtaking (Low level)

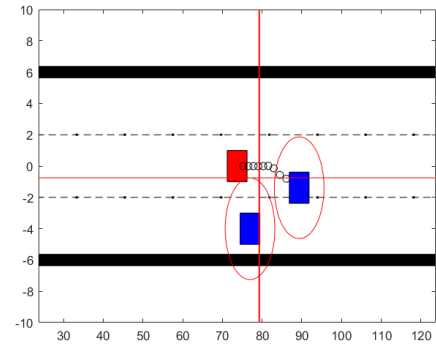


(b) Horizontal constraint (Low level)

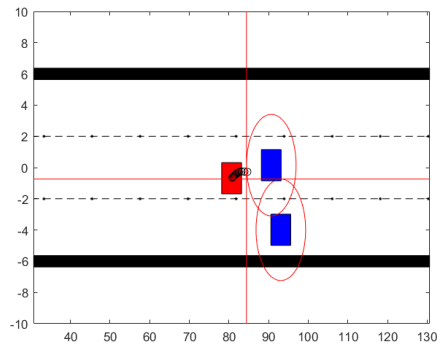
Figure 3.16: TV switches to the middle lane



(a) TV switches lane (High level)



(b) TV stops (Low level)



(c) End of the simulation (Low level)

Figure 3.17: Last steps of the simulation





## Chapter 4

# Improvements and further development

Although the simulation is now successfully working there a couple of improvements which could be implemented in the future. The first improvement which could be implemented in the future is roads with curvature, which would make the algorithm actually usable in a real world situation, since most of the roads in the real world aren't straight either. Another improvement which could be implemented are specific situation constraints. For example in a situation where a TV stops the ellipse constraint could be greatly reduced since the EV comes to a stop. Specific constraints would make the algorithm much more applicable to a real world situation. A more realistic real world simulation could challenge the algorithm. Also more complex environment with multiple vehicles could be added to see how the algorithm solves those problems.



# List of Figures

3.1	Following the reference . . . . .	20
3.2	Overtaking . . . . .	21
3.3	Switching back to the original lane . . . . .	22
3.4	Initial position (High level) . . . . .	23
3.5	Position after accelerating (Low level) . . . . .	23
3.6	Position after accelerating (High level) . . . . .	24
3.7	Starting positions of the high and low level . . . . .	24
3.8	Overtaking maneuver . . . . .	25
3.9	Occurrence of the error . . . . .	25
3.10	Starting position (High level) . . . . .	26
3.11	Constraint applied (Low level) . . . . .	26
3.12	Position after accelerating with low level constraints (High level) . . .	27
3.13	EV comes to a stop (Low level) . . . . .	27
3.14	EV moves forward (Low level) . . . . .	28
3.15	Starting positions . . . . .	28
3.16	TV switches to the middle lane . . . . .	29
3.17	Last steps of the simulation . . . . .	29



# List of Tables



# Bibliography

- [BBL21] Tommaso Benciolini, Tim Brüdigam, and Marion Leibold. Multi-stage stochastic model predictive control for urban automated driving. *arXiv preprint arXiv:2107.00529*, 2021.
- [BGW14] Andrea Boccia, Lars Grüne, and Karl Worthmann. Stability and feasibility of state constrained mpc without stabilizing terminal constraints. *Systems & control letters*, 72:14–21, 2014.
- [BOWL21] Tim Brüdigam, Michael Olbrich, Dirk Wollherr, and Marion Leibold. Stochastic model predictive control with a safety guarantee for automated driving. *IEEE Transactions on Intelligent Vehicles*, 2021. doi:10.1109/TIV.2021.3074645.
- [GP17] Lars Grüne and Jürgen Pannek. Nonlinear model predictive control. In *Nonlinear Model Predictive Control*. Springer, 2017.
- [KJK<sup>+</sup>13] Junsoo Kim, Kichun Jo, Dongchul Kim, Keonyup Chu, and Myoungcho Sunwoo. Behavior and path planning algorithm of autonomous vehicle in structured environments. *IFAC Proceedings Volumes*, 46(10):36–41, 2013.
- [QNdLFM16] Xiangjun Qian, Iñaki Navarro, Arnaud de La Fortelle, and Fabien Moutarde. Motion planning for urban autonomous driving using Bézier curves and mpc. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 826–833, 2016. doi:10.1109/ITSC.2016.7795651.





# License

This work is licensed under the Creative Commons Attribution 3.0 Germany License. To view a copy of this license, visit <http://creativecommons.org> or send a letter to Creative Commons, 171 Second Street, Suite 300, San Francisco, California 94105, USA.