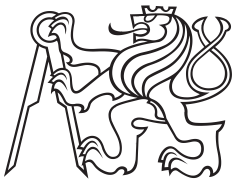
Master Thesis



Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Optimal Planning and Control of Vehicle Dynamics

Bc. Petr Turnovec

Supervisor: Ing. David Vošahlík
Field of study: Cybernetics and Robotics
May 2021

MASTER'S THESIS ASSIGNMENT

I. Personal and study details

Student's name: **Turnovec Petr** Personal ID number: **459919**

Faculty / Institute: **Faculty of Electrical Engineering**

Department / Institute: **Department of Control Engineering**

Study program: **Cybernetics and Robotics**

Branch of study: **Cybernetics and Robotics**

II. Master's thesis details

Master's thesis title in English:

Optimal planning and control of vehicle dynamics

Master's thesis title in Czech:

Optimální plánování a řízení dynamiky vozidla

Guidelines:

The goal of this master's thesis will be to evaluate performance of two optimization-based methods - model predictive control (MPC) and minimum violation planning (MVP). The work will focus on Vehicle dynamics control/planning problems solution achieving safety and overall efficiency of the system. Problems will range from vehicle dynamics control, which is domain of MPC, to the purely planning tasks. Constraints of planning, which is domain of MVP, shall be in form of logical statements (LTL formulas). The thesis will be worked out in cooperation with industry - Garrett motion.
The thesis will compose of following steps:
1. Develop/adopt non-linear verification mathematical model
2. Develop design models suitable for MPC (different complexity levels)
3. Implement LTL (linear temporal logic) formulas as MPC constraints
4. Develop design models suitable for MVP (different complexity levels)
5. Implement both control algorithms
6. Use Garrett tools for implementation
7. Validate and compare both MVP and MPC algorithms in simulations

Bibliography / sources:

[1] Dieter Schramm, Manfred Hiller, Roberto Bardini – Vehicle Dynamics – Duisburg 2014
[2] Hans B. Pacejka - Tire and Vehicle Dynamics – The Netherlands 2012
[3] B. Brito, B. Floor, L.Ferranti, J. Alonso-Mora - Model Predictive Contouring Control for Collision Avoidance in Unstructured Dynamic Environments - IEEE Robotics and Automation Letters 2019
[4] T. Wongpiromsarn, K. Slutsky, E. Frazzoli, U. Topcu - Minimum-violation planning for autonomous systems: Theoretical and practical considerations - arXiv 2020

Name and workplace of master's thesis supervisor:

Ing. David Vošahlík, Department of Control Engineering, FEE

Name and workplace of second master's thesis supervisor or consultant:

Ing. Tomáš Haniš, Ph.D., Department of Control Engineering, FEE

Date of master's thesis assignment: **25.01.2021** Deadline for master's thesis submission: **21.05.2021**

Assignment valid until:
by the end of summer semester 2021/2022

Ing. David Vošahlík	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
Supervisor's signature	Head of department's signature	Dean's signature

III. Assignment receipt

The student acknowledges that the master's thesis is an individual work. The student must produce his thesis without the assistance of others, with the exception of provided consultations. Within the master's thesis, the author must state the names of consultants and include a list of references.

.	
Date of assignment receipt	Student's signature

Acknowledgements

First, I would like to thank my supervisor Ing. David Vošahlík for his valuable advice and supportive supervision of my work.

Great thanks also belong to Smart Driving Solution team at Department of Control Engineering. I namely thank Ing. Tomáš Haniš, Ph.D. and Bc. Marek Boháč.

My thanks also belongs to Garrett Motion Czech Republic s.r.o, namely to Ing. Jaroslav Pekař, Ph.D. for his helpful advice.

Last, yet the most, I would like to thank my family for their support.

Declaration

I hereby declare that this master's thesis was finished on my own and that I have cited all the used information sources in compliance with the Methodical guideline on the observance of ethical principles in the preparation of university graduate thesis.

In Prague, 20 May 2021

Abstract

The master thesis deals with comparison of model predictive control and minimum-violation planning for vehicle trajectory optimal planning and control problems. Planning problems are defined using linear temporal logic formulas that are easy to formulate by a designer. However, its transformation to model predictive control constraints introduced is highly problem-specific. First, mathematical models of vehicle individual components are presented. Then, a high-fidelity vehicle dynamics model is integrated. Next, nonlinear model predictive control method is briefly described and vehicle design models suitable for control by the method are derived. The method solution for a nonconvex environment with obstacles is proposed. Further, an introduction to minimum-violation planning method follows. Vehicle design models suitable for control by minimum-violation planning are derived and modifications to the method for dynamic systems with a high number of states are proposed. Finally, results of the methods are presented and the methods comparison for simple demo scenarios is given.

Keywords: control of vehicle dynamics, optimal trajectory planning, model predictive control, minimum-violation planning, environment with obstacles

Supervisor: Ing. David Vošahlík

Abstrakt

Tato práce porovnává metody model predictive control a minimum-violation planning pro optimální plánování trajektorie vozidla a jeho řízení. Problémy plánování jsou definovány pomocí výroků lineární temporální logiky, které jsou snadné na formulaci. Jejich uvedená tranformace na omezení pro model predictive control je ale závislá na konkrétním problému. Nejprve jsou uvedeny matematické modely jednotlivých komponent vozidla a poté je sestaven dynamický model vozidla pro verifikaci. Dále je stručně popsána nelineární varianta metody model predictive control, jsou odvozeny zjednodušené modely vozidla vhodné pro řízení touto metodou a je navženo řešení pro nekonvexní prostředí s překážkami. Následuje představení metody minimum-violation planning, odvození zjednodušených modelů vozidla vhodných pro minimum-violation planning, a jsou navženy modifikace této metody pro dynamické systémy s vyšším počtem stavů. Nakonec jsou podány výsledky obou metod a je uvedeno porovnání těchto metod na jednoduchých ukázkových scénářích.

Klíčová slova: řízení dynamiky vozidla, optimální plánování trajektorie, model predictive control, minimum-violation planning, prostředí s překážkami

Překlad názvu: Optimální plánování a řízení dynamiky vozidla

Contents

Figures

Chapter 1

Introduction

In the last decade, the development of autonomous vehicle industry experiences a boom. Many companies and institutions are trying to replace a driver by designing algorithms and creating more or less vehicles controlled by artificial intelligence. In some cases, partial replacement of a driver or an assistance provided to the driver is introduced. On the other side, in some cases, fully self-driving cars are designed.

Systems that are fully in charge of vehicle control can be assigned to the latter group. These are complex systems that have to

(i) ■ **recognize** the vehicle surrounding environment,

(ii) ■ process information, **make a decision**, plan a path,

(iii) ■ **ensure proper execution** of the decision.

The part (i) is domain of computer vision. Next part (ii) can be provided by one of many decision making algorithms. For example, it can be based on machine learning, neural networks, Markov decision processes, control theory, ..., or whatever meaningful planning algorithm. The last part (iii) is mainly a domain of control theory (feedback, estimation processes).

Assistance systems ensuring safety or systems improving vehicle behavior are, for example,

- **Anti-lock braking system (ABS)**
  A wheel brake can cause wheel blocking and thus, braking moment decrease. Anti-look braking system is the usual system that prevents too high undesirable slip of the tires in situations when the car is braking hard.

- **Electronic stability program (ESP)**
  Electronic stability program handles the lateral stabilization of the vehicle. The ESP aims to stabilize the yaw rate (angular speed around the vehicle center of gravity) in critical situations using brakes. Stabilization is ensured by brake control at individual wheels and appropriate reduction of the braking action on the particular wheel.

- **Traction control (TC)**
  Traction control technology has the same purpose as ABS. However, it can control the drive torque and therefore it prevents undesirable slip of the tires when the car is accelerating.

- **Torque vectoring (TV)**
  Torque vectoring system is similar to ESP. The two are in a similar relationship as ABS and TC. Torque vectoring actively uses torques on individual wheels to ensure yaw stabilization.

These systems are mainly in charge of vehicle dynamics. However, the responsibility for driving still goes to the driver.

The thesis will discuss items (ii) - (iii). Therefore, it is assumed that the surrounding environment is known. Static or moving obstacles are identified and their positions are accurately given as well as the vehicle position in the environment. The aim is to plan a path (make a decision) and go along with that path or replan the path (feedback). On the other hand, the stated assistance algorithms (ABS, ESP, TC, TV) will not be discussed at all as it is not the thesis objective and their functionality can be eventually included directly in the decision making algorithm.

Goal of the thesis is to compare two optimization-based algorithms - model predictive control and minimum-violation planning - for planning tasks and for vehicle dynamics control. Model predictive control is a widely used method in control theory. It is exploited mainly for control of dynamic systems. The thesis tries to implement model predictive control also for planning in the environment with obstacles. Minimum violation planning is a sampling-based algorithm and its domain is primarily path planning. The method is implemented to planning for a large dynamic system (vehicle) and so under a lot of dynamic constraints in the thesis.

Both methods should fulfill requirements mainly regarding safety (obstacle avoidance) and other requirements like travel time or fuel consumption minimization. Results are obtained from simulations. For verification, high-fidelity single-track model of the vehicle is used.

Problem of planning in the environment with obstacles, which will be discussed in the thesis, is introduced and test scenarios for verification purposes are given in chapter 2. High-fidelity single-track model of a vehicle is presented and its components are described in the next chapter 3. Following chapter 4 focuses on model predictive control. At first, an introduction to nonlinear model predictive control (NMPC) is stated, then design models of the vehicle suitable for control by NMPC are derived, and finally, solution by NMPC to the analyzed problem is discussed and proposed. Results of NMPC are shown in the last section of the chapter. Next chapter 5 is about minimum-violation planning (MVP). At first, an introduction to MVP is presented, then design models of the vehicle suitable for control by MVP are provided, and further modifications to MVP are proposed to deal with large systems with dynamics constraints. Finally, feedback for MVP is discussed. Furthermore, the results of MVP are given for open loop structure and for feedback structure. Chapter 6 presents a comparison of both methods NMPC and MVP. The thesis is summarized and concluded in the last chapter 7.

Chapter 2

Problem Formulation, Test Scenarios

The problem formulation is not written down rigorously but rather sketched. Differences in the methods MPC and MVP intended for solution of this problem and their different domains are the main reason for this. The problem analyzed in the thesis is following.

A dynamic system - vehicle - is given, which will be controlled. A goal is to steer the vehicle from the start state to the goal state (or position). Thus, the problem is trajectory generation, not trajectory tracking. Environment contains obstacles and these can be static or moving. Furthermore, the obstacles can be impassable like buildings, other cars, pedestrians, or the obstacle can be passable, but it is not suitable and advantageous to go through it, like pavement, grass.

Moreover, requirements which a trajectory should fulfill can be given. Thus, for example, (except the obstacle avoidance) time or consumption minimization, or smooth ride ensuring (close consecutive steer inputs) and etc. States of the dynamic system can be also constrained - for example, maximal or minimal speed.

It is assumed that obstacles have been already identified and their positions are known. The obstacles are considered as polygons and their vertices in state space. Thus, obstacles with round shapes were transformed to polygons in advance. A vehicle is modeled as a spatial or rather planar object in the following chapter, in spite of that, the controlled vehicle is assumed to be a mass point regarding the collision avoidance or in case of any other constraint fulfillment check. The surrounding obstacles can be inflated and

other constraints reformulated to capture the fact that the vehicle is not a mass point if it is needed.

The problem is illustrated by the following scenarios. The depicted situations will serve as test scenarios for the algorithms. The first scenario (Fig. 2.1) is environment without any static or moving obstacles, only the vehicle dynamics is controlled, and requirements for optimization of some variables are presented. The second scenario (Fig. 2.2) is environment with static obstacles (red objects) and possibly also with moving obstacles (the gray one). The last situation is slightly different. It is an overtaking scenario (Fig. 2.3), where the goal is to overtake the other car, if it is advantageous from the optimization point of view, while staying on the road and tracking the center line of the right lane. Here, the problem is close to trajectory tracking (or better path tracking).

Constraints, which violate the convexity required for the other state variables (like velocity, yaw rate, state of charge) will be briefly discussed in the comparison chapter 6.

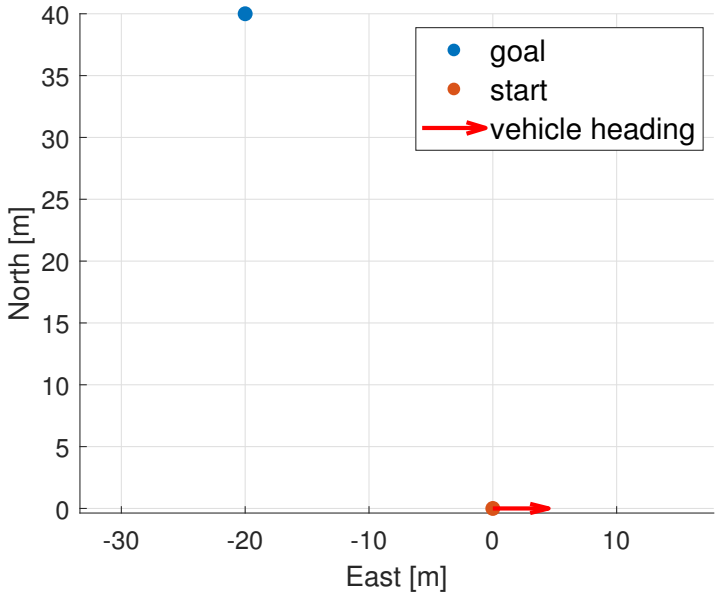


Figure 2.1: Scenario 1 - environment without static obstacles or moving obstacles, only dynamics constraints.

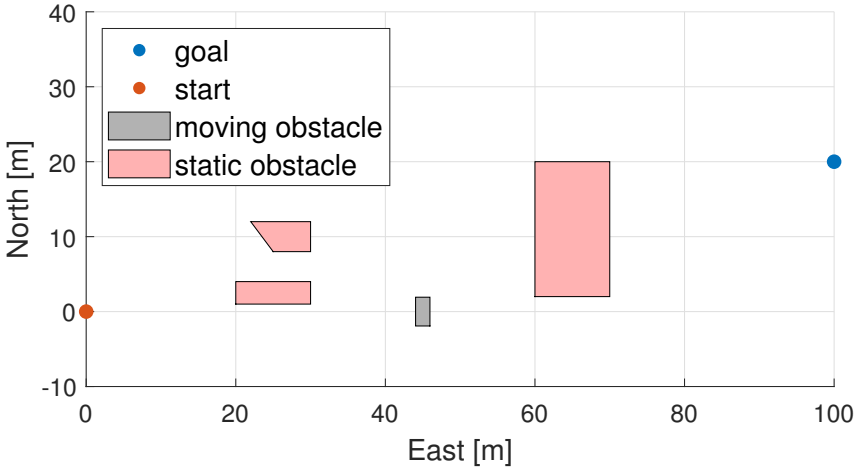


Figure 2.2: Scenario 2 - environment with static obstacles and/or moving obstacles.

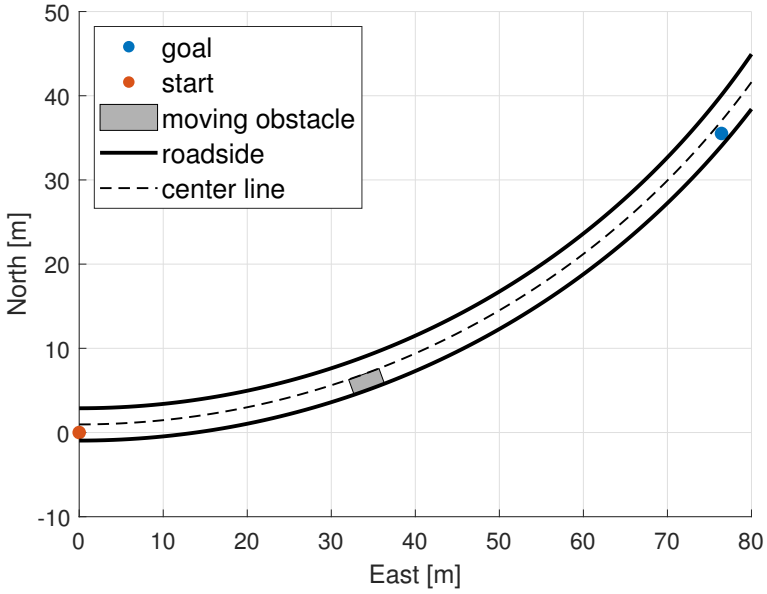


Figure 2.3: Scenario 3 - environment with moving obstacle, tracking path (position points) - overtaking.

Chapter 3

Vehicle Modeling

The high fidelity model of the vehicle (plant model) which will be used through the work is described in this chapter. The model would serve in the thesis as a verification tool because all results are acquired only in simulations. On the following lines, a single-track model of the vehicle is derived. Single track model has considerably fewer states than more complex twin track models [20]. It is given not only by "tracks" reduction, but mainly by lifting, rolling and pitching motion neglect. However, it is sufficient and precise enough for purpose of the thesis. For the single-track model used here, the front wheels are represented as a single one as well as rear ones. Both wheels (front and rear) of the vehicle are driven by its own electric motor. These motors are powered by one battery. Battery can be charged during the ride as regenerative braking by the motors is considered. Single track model is shown in the figure 3.1.

The single-track vehicle dynamics model was adopted with modifications from [5] (similar models are described in [14] and [12]), tire model was derived according to [16], electric motor model was adopted from [10] and battery model from [3]. Implementation of electric motor model and battery model according to stated papers used by Garrett motion were adopted and modified. Various vehicle modeling techniques and its extensive description can be found in literature [6], [13] and [20], where single-track and twin-track models are described and the modeling of lifting, rolling and pitching motion is given.

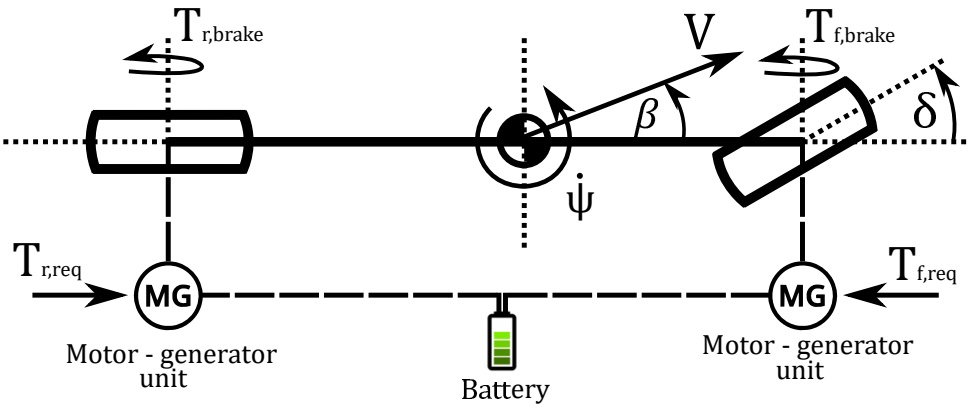


Figure 3.1: High fidelity single track model.

3.1 Vehicle components

Vehicle is divided into various components for clarity. This means easy composition of different models and different configurations, if needed (eg., twin track with 4 motors, etc.). The main advantage of this approach is easy components re-usability. Quite naturally, the vehicle single track model was divided into four components:

- electric motor,
- battery,
- wheel including tire,
- vehicle dynamics.

These components will be described in this section. For each component, there will be given table enumerating inputs, states and outputs of the state space representation in a unified manner.

3.1.1 Electric motor model

Electric motor (EM) is modeled as a system that consists of 2 states, 2 inputs and 2 outputs according to the article [10].

Variable	Description	Units
Inputs		
T_{req}	Requested torque	Nm
ω_{in}	Shaft speed	rad/s
States		
T_{int}	EM torque	Nm
ω_{em}	EM speed	rad/s
Outputs		
T_{em}	EM torque out	Nm
P_{em}	EM electric power	W

Table 3.1: Electric motor - system representation.

The state equations are

$$\dot{T}_{int} = -\frac{1}{t_T}(T_{int} - \min(\max(T_{req}, T_{int,min}), T_{int,max}), \tag{3.1}$$

$$\dot{\omega}_{em} = -\frac{1}{t_\omega}(\omega_{em} - \omega_{in}). \tag{3.2}$$

The first state T_{int} represents the mechanical torque produced by the motor. Time constant t_T in Eq. 3.1 determine the transition speed of input torque request T_{req} tracking. The input torque request is limited to be in interval $\langle T_{int,min}, T_{int,max}\rangle$. The other state ω_{em} is angular speed of the motor. Eq. 3.2 models shaft and motor speed differentiation with time constant t_ω. The equation represents the fact that the connection is not fixed.

Outputs of the system are drive torque T_{em} for wheel and electric power P_{em} consumed or produced (depends on the sign),

$$T_{em} = T_{int} r_\omega - J_{em} r_\omega^2 (-\frac{1}{t_\omega}(\omega_{em} - \omega_{in})), \tag{3.3}$$

$$P_{em} = T_{int} r_\omega \omega_{in} \eta_{-1}(T_{int}, \omega_{in}). \tag{3.4}$$

Note in Eq. 3.3 the term $-\frac{1}{t_\omega}(\omega_{em} - \omega_{in})$ which means EM acceleration. Therefore, the second term in the equation represents the torque "taken" by EM. In both of these output equations r_ω represents gear ratio. In the second output equation 3.4 efficiency (inverse efficiency) $\eta_{-1}(T_{int}, \omega_{in})$ is taken into account. The efficiency is modeled as a function dependent on the power consumed or produced by the motor. The efficiency models losses of electric motor. When the motor is powered, the inverse efficiency $\eta_{-1}(T_{int}, \omega_{in})$ is greater than one (the supplied power is greater than efficient power). On the other hand, when the motor generates power, the inverse efficiency $\eta_{-1}(T_{int}, \omega_{in})$ is smaller than one. Electric motor (EM) is powered by battery. It is important to say here that consumed or produced electric power should respect battery limitations (battery power is limited). This is ensured by appropriate control system.

3.1.2 Battery model

Battery model was developed and modified according to [3]. The paper models a battery as an electro-thermal system. The thermal part is neglected in the thesis as the thermal behavior modeling is not in interest of the thesis. It can be done because thermal management is assumed and the thermal condition has negligible impact on usual modes of the battery and their significance to the electrical parts of the battery grows only in marginal conditions. The resulting electrical model is described by circuit in the following figure 3.2.

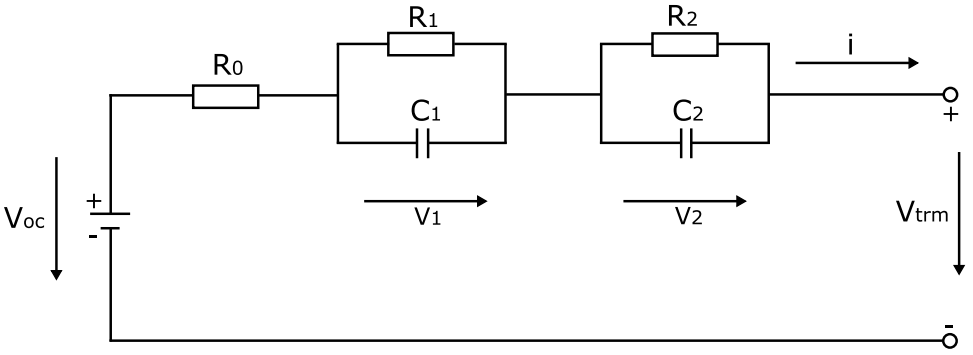


Figure 3.2: Battery circuit equivalent scheme.

The circuit consists of one DC source, 2 RC pairs and one separate resistor. According to the figure, three state equations 3.5, 3.6 and 3.7 are written down, where the third one describes the state of charge (SoC) of the battery,

$$\dot{v}_1 = -\frac{1}{R_1 C_1} v_1 + \frac{1}{C_1} i, \tag{3.5}$$

$$\dot{v}_2 = -\frac{1}{R_2 C_2} v_2 + \frac{1}{C_2} i, \tag{3.6}$$

$$\dot{SoC} = -\frac{100}{3600C} i. \tag{3.7}$$

In the equations, R_1, R_2 denotes to resistors, C_1, C_2 to capacitors, C to battery capacity and i to the circuit current. The system representation is in the table 3.2. Because the important state is the state of charge, it is the only output. The input to the system is power P_{trm}, which is supplied by the battery or power for the battery charging, depending on the sign of the input.

Variable	Description	Units
Inputs		
P_{trm}	Battery terminal power	W
States		
SoC	State of charge	%
v_1	Voltage across the R_1C_1 pair	V
v_2	Voltage across the R_2C_2 pair	V
Outputs		
SoC	State of charge	%

Table 3.2: Battery - system representation.

Circuit current occurring in the state equations is calculated in Eq. 3.8, where terminal voltage V_{trm} is computed in Eq. 3.9. The equation 3.10 is obtained by current substitution from Eq. 3.8. In these equations, V_{oc} represents open circuit voltage, which is function of SoC as well as R_0 (Eq. 3.11, 3.12). Example of such a dependence of V_{oc} on SoC is in the figure 3.3.

$$i = \frac{P_{trm}}{V_{trm}}, \tag{3.8}$$

$$V_{trm} = V_{oc} - v_1 - v_2 - R_0 i, \tag{3.9}$$

$$V_{trm} = \frac{V_{oc} - v_1 - v_2 + \sqrt{(V_{oc} - v_1 - v_2)^2 - 4R_0P_{trm}}}{2}, \tag{3.10}$$

$$V_{oc} = f_{ocv}(SoC) \overset{eg.}{=} 25 \cdot 10^{-4} \left(SoC - 55\right)^3 + 360, \tag{3.11}$$

$$R_0 = f_{R0}(SoC) \overset{eg.}{=} 2 \cdot 10^{-4} SoC + 9 \cdot 10^{-3}. \tag{3.12}$$

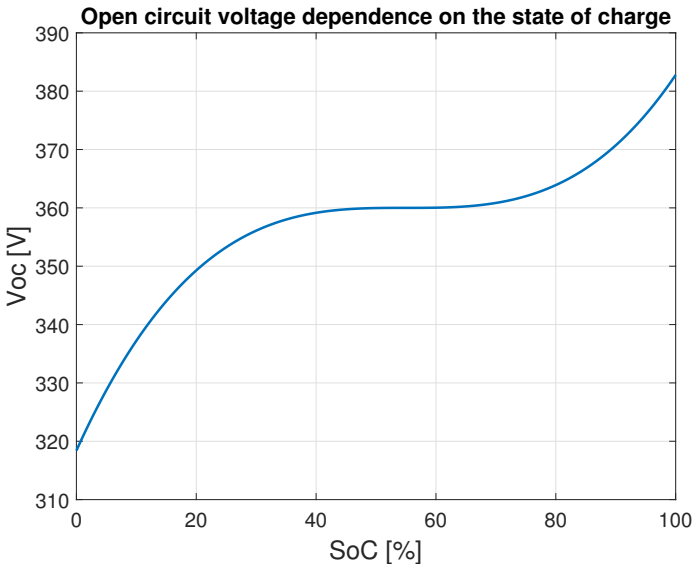
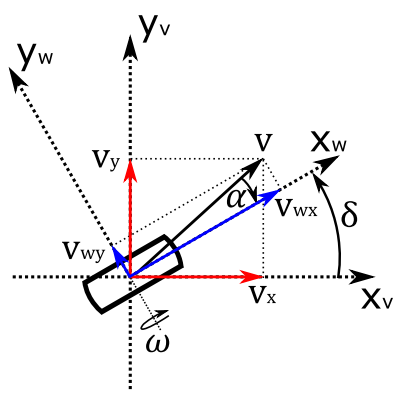


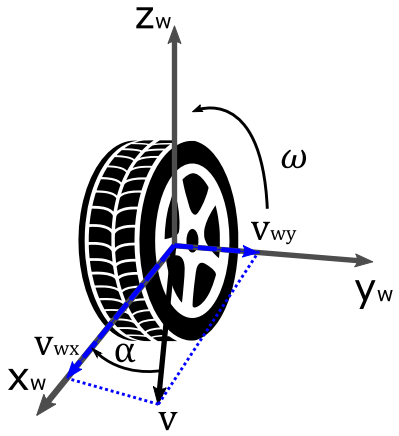
Figure 3.3: Open circuit voltage dependence on the state of charge. Example given for parameters used in the thesis.

3.1.3 Wheel model

The next component which will be described is a wheel including a tire. Coordinate systems (CS) shown in the figure 3.4 are used. Figure on the left 3.4a shows overall situation. Tire is steered by steering angle δ. Coordinate frame with subscript v is bounded with vehicle coordinates originated at the wheel pivot point. Coordinates with subscript w are bounded with a wheel. Axis x_w points to the front of the tire, y_w axis originates in the tire center of rotation and points to the left.



(a) : Wheel (w) vs. vehicle (v) coordinate frame.



(b) : Wheel and its coordinate frame.

Figure 3.4: Wheel coordinate system.

Tire system representation is given in table 3.3. Inputs to the system are

Variable	Description	Units
Inputs		
δ	Steering angle	rad
T_w	Wheel drive torque	Nm
$T_{w,br}$	Wheel brake torque	Nm
v_x	Wheel translation velocity in vehicle CS in direction x	m/s
v_y	Wheel translation velocity in vehicle CS in direction y	m/s
F_z	Wheel vertical load	N
States		
ω	Wheel angular velocity	rad/s
Outputs		
F_x	Longitudinal force generated by tire, in vehicle CS	N
F_y	Lateral force generated by tire, in vehicle CS	N
ω	Wheel angular velocity	rad/s

Table 3.3: Wheel system representation.

steer angle δ, drive and brake torque T_ω and $T_{\omega,br}$, wheel translation velocity in vehicle CS in direction x - v_x and y - v_y respectively and wheel vertical load F_z. According to the given figure 3.4a velocities are transformed to the tire coordinate frame by the following equations

$$v_{wx} = v_x \cos\delta + v_y \sin\delta, \tag{3.13}$$
$$v_{wy} = -v_x \sin\delta + v_y \cos\delta. \tag{3.14}$$

Tire is the only contact point of the vehicle with the road and therefore, it generates forces that act on the vehicle. The goal is to get these forces now. One of the most used approaches to do that is by using so-called Pacejka magic formula designed by Hans B. Pacejka [16]. Various other techniques for tire modeling are described for example in [9]. By this Pacejka approach it is possible to calculate the longitudinal and lateral forces that the tire generates. For this, slip ratio and side slip angle of the tire have to be computed first.

Tire translation velocity is not equal to the tire peripheral speed. This difference describes slip ratio λ by equation

$$\lambda = \frac{\omega R - v_{wx}}{\max(\omega R, |v_{wx}|)}, \tag{3.15}$$

where ω is angular speed, R is radius and v_{wx} is the translation velocity in x direction of the tire.

Moreover, the wheel velocity direction is not the same as its orientation. The fact describes side slip angle which is given as (convention where positive rotations are anticlockwise is used throughout the work)

$$\alpha = -\arctan\frac{v_{wy}}{v_{wx}} \tag{3.16}$$

Now Pacejka magic formulas are introduced to compute the longitudinal and lateral forces of the tire,

$$F_{wx}(\lambda) = F_z D_x \sin(C_x \arctan(B_x\lambda - E_x(B_x\lambda - \arctan(B_x\lambda)))), \tag{3.17}$$
$$F_{wy}(\alpha) = F_z D_y \sin(C_y \arctan(B_y\alpha - E_y(B_y\alpha - \arctan(B_y\alpha)))). \tag{3.18}$$

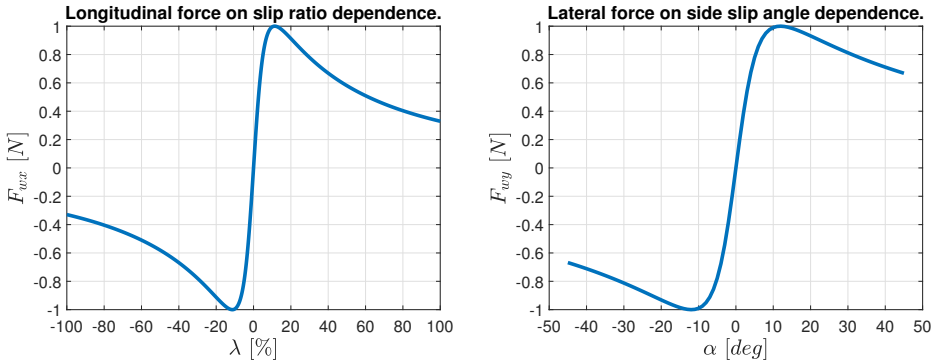
In these equations, B_x, C_x, D_x, E_x and B_y, C_y, D_y, E_y are the longitudinal and lateral coefficients of the given tire. Example of dependence of the longitudinal force on the slip ratio and lateral force on the slip angle is shown in the figure 3.5. Finally, these forces have to be transformed back to the vehicle coordinate frame (Fig. 3.6) by transformation

$$F_x = F_{wx}\cos\delta - F_{wy}\sin\delta, \tag{3.19}$$
$$F_y = F_{wx}\sin\delta + F_{wy}\cos\delta. \tag{3.20}$$

The wheel system has one state - wheel angular speed ω, whose dynamics is defined as

$$\dot{\omega} = \frac{1}{J}(T_w - F_{wx}R - sgn(\omega)T_{w,br}), \tag{3.21}$$

where J is the moment of inertia of the wheel.



(a) : Longitudinal force. **(b) :** Lateral force.

Figure 3.5: Pacejka magic formula. Generated forces for tire parameters used in the thesis (appendix B.3) with $F_z = 1\,\mathrm{N}$.

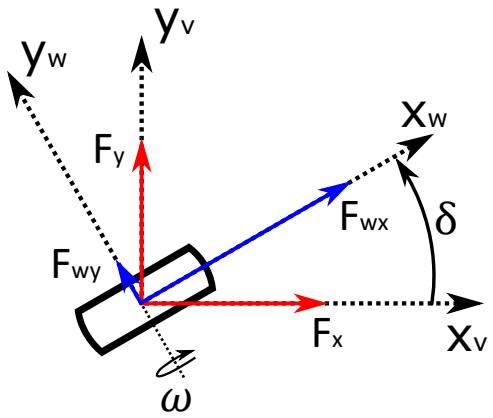


Figure 3.6: Transformation of tire forces back to the vehicle coordinate frame.

3.1.4 Vehicle dynamics model

Finally, the vehicle dynamics will be described. For this purpose single-track model of the vehicle was adopted (see [5], [14], [12]). For a single-track model, the front wheels are represented as a single one as well as rear ones. Moreover, the pitching, rolling and lifting motions are neglected. However, single-track model captures quite well the dynamics of the vehicle. Mainly the variables

we are interested in are modeled - position, yaw angle, velocity, yaw rate. Finally, the model provides enough precision for thesis purposes.

Single-track is shown in the figure 3.7. Vehicle x axis goes from the center of gravity (CoG) towards the front tire, axis y from CoG towards the left side of the vehicle and axis z according to right-handed orientation, upwards. Inputs to the model are forces generated by the front and rear tires F_{xf}, F_{yf} and F_{xr}, F_{yr}. Vehicle velocity v direction is given by side slip angle β. Vehicle yaw rate around the z axis is $\dot{\psi}$. Longitudinal distance of the front, resp. rear axle from CoG is l_f, resp. l_r. Torque around the z axis is denoted as M_z.

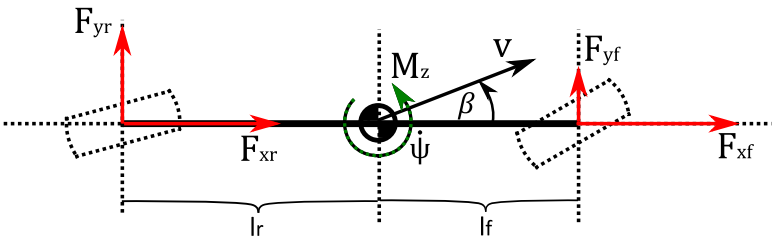


Figure 3.7: Vehicle forces.

Now the following forces and moment equations can be written down,

$$F_x = F_{xf} + F_{xr} - F_{aero} - F_{slope}, \tag{3.22}$$
$$F_y = F_{yf} + F_{yr}, \tag{3.23}$$
$$M_z = l_f F_{yf} - l_r F_{yr}, \tag{3.24}$$

where F_{aero} and F_{slope}, which are calculated as

$$F_{aero} = \frac{1}{2}\rho_a c_d A_f v^2, \tag{3.25}$$
$$F_{slope} = mg \sin \alpha_{inc}, \tag{3.26}$$

are not generated through the tires, but act directly on the vehicle. In Eq. 3.25, 3.26 ρ_a stands for air density, c_d for drag coefficient, A_f for vehicle frontal area, m for vehicle mass and α_{inc} is the inclination angle of the road.

Three states describing the vehicle dynamics are given as (for a detailed derivation see eg. [22])

$$\dot{v} = \frac{1}{m}(\sin(\beta)F_y + \cos(\beta)F_x), \tag{3.27}$$
$$\dot{\beta} = -\dot{\psi} + \frac{1}{mv}(\cos(\beta)F_y - \sin(\beta)F_x), \tag{3.28}$$
$$\ddot{\psi} = \frac{1}{I_z}M_z, \tag{3.29}$$

where I_z is the vehicle moment of inertia with respect to z axis.

Moreover, it is needed to describe the position and yaw angle of the vehicle. For this purpose, additional equations are needed

$$\dot{e} = v\cos(\beta + \psi), \tag{3.30}$$
$$\dot{n} = v\sin(\beta + \psi), \tag{3.31}$$
$$\dot{\psi} = \dot{\psi}, \tag{3.32}$$

where e is the east position and n is the north position.

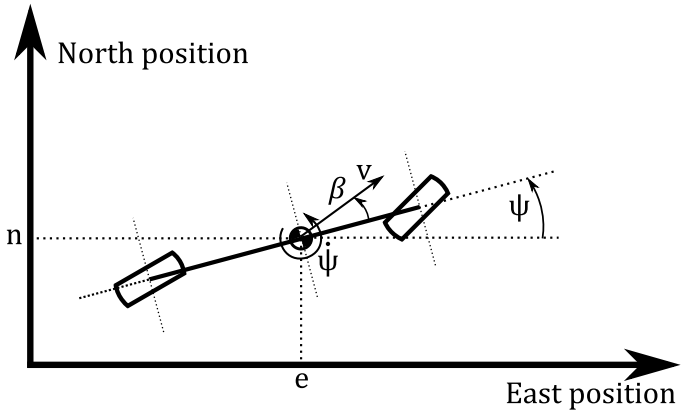


Figure 3.8: Vehicle position and orientation.

Previously stated inputs and states summarizes table 3.4. There are given additional outputs in the table. These outputs are inputs for the tires. Velocities of the front and rear tire can be calculated according to figure 3.9 as

$$v_{xf} = v\cos\beta, \tag{3.33}$$
$$v_{xr} = v\cos\beta, \tag{3.34}$$
$$v_{yf} = v\sin\beta + l_f\dot{\psi}, \tag{3.35}$$
$$v_{yr} = v\sin\beta - l_r\dot{\psi}. \tag{3.36}$$

And the forces loading these tires are given by the equations

$$F_{zf} = mg\frac{l_r}{l_f + l_r}, \tag{3.37}$$
$$F_{zr} = mg\frac{l_f}{l_f + l_r}. \tag{3.38}$$

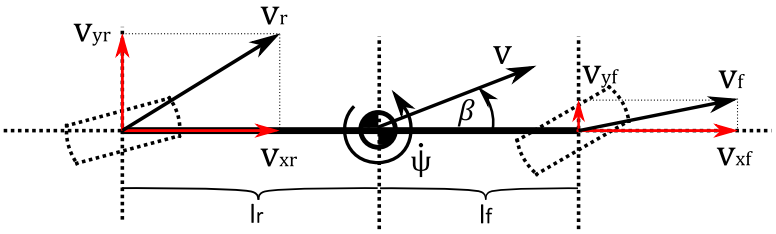


Figure 3.9: Vehicle velocities.

Variable	Description	Units
Inputs		
α_{inc}	Inclination angle	rad
F_{xf}	Longitudinal force generated by front tire, in vehicle CS	N
F_{xr}	Longitudinal force generated by rear tire, in vehicle CS	N
F_{yf}	Lateral force generated by front tire, in vehicle CS	N
F_{yr}	Lateral force generated by rear tire, in vehicle CS	N
States		
e	East position	m
n	North position	m
ψ	Yaw angle of vehicle	rad
v	Velocity	m/s
β	Side slip angle	rad
$\dot{\psi}$	Yaw rate	rad/s
Outputs		
States	All the states are outputs	-
v_{xf}	Front tire longitudinal translation velocity, in vehicle CS	m/s
v_{xr}	Rear tire longitudinal translation velocity, in vehicle CS	m/s
v_{yf}	Front tire lateral translation velocity, in vehicle CS	m/s
v_{yr}	Rear tire lateral translation velocity, in vehicle CS	m/s
F_{zf}	Front tire vertical load, in vehicle CS	N
F_{zr}	Rear tire vertical load, in vehicle CS	N

Table 3.4: Vehicle dynamics system representation.

3.2 High-fidelity single-track model

Now it is possible to compose a complex high-fidelity single-track model, which consists of one battery and two tires, each of them driven by its own electric motor as in the figure 3.1 at the beginning of the chapter.

Inputs to individual components will be divided into two categories - manipulated inputs and disturbance inputs. Inputs that can be controlled

(manipulated) belong to the first category, whereas in the latter one are inputs given by outputs of the other components or given externally. These inputs are marked in the table 3.5. This table gives overview how the overall model is composed from the components.

Input	Type	Description
Front EM		
T_{req_f}	Manipulated	-
ω_{in_f}	Disturbance	Proportionally given by front tire ω_f
Rear EM		
T_{req_r}	Manipulated	-
ω_{in_r}	Disturbance	Proportionally given by rear tire ω_r
Battery		
P_{trm}	Disturbance	Given as $P_{trm} = P_{em_f} + P_{em_r}$
Front tire		
δ_f	Manipulated	-
T_{ω_f}	Disturbance	Given by front EM out torque T_{em_f}
T_{ω,br_f}	Manipulated	-
v_{x_f}	Disturbance	Given by vehicle v_{xf} output
v_{y_f}	Disturbance	Given by vehicle v_{yf} output
F_{z_f}	Disturbance	Given by vehicle F_{zf} output
Rear tire		
δ_r	Manipulated	-
T_{ω_r}	Disturbance	Given by rear EM out torque T_{em_r}
T_{ω,br_r}	Manipulated	-
v_{x_r}	Disturbance	Given by vehicle v_{xr} output
v_{y_r}	Disturbance	Given by vehicle v_{yr} output
F_{z_r}	Disturbance	Given by vehicle F_{zr} output
Vehicle dyn		
α_{inc}	Disturbance	Given by the road inclination
F_{xf}	Disturbance	Given by front tire output F_{x_f}
F_{yf}	Disturbance	Given by front tire output F_{y_f}
F_{xr}	Disturbance	Given by rear tire output F_{x_r}
F_{yr}	Disturbance	Given by rear tire output F_{y_r}

Table 3.5: Classification of inputs to vehicle components.

The high-fidelity model was developed with 6 inputs that can be controlled - δ_f, δ_r, T_{req_f}, T_{req_r}, T_{ω,br_f}, T_{ω,br_r} - and 15 states - T_{int_f}, T_{int_r}, ω_{em_f}, ω_{em_r}, SoC, v_1, v_2, ω_f, ω_r, e, n, ψ, v, β, $\dot{\psi}$.

However, for both NMPC and MVP algorithms that are to be implemented **only front steering angle δ_f, drive torque T_{req_f} and T_{ω,br_f} will be controlled**. Inputs δ_r, T_{req_r} and T_{ω,br_r} will be zeros. Presence of these inputs

will be used in further work.

All used parameters for individual components and their values can be seen in appendix B.

3.3 Model Validation

Now the derived high fidelity model should be validated to ensure real vehicle reality matching. For this purpose, the reader is referred to publications where validation was done. Single-track vehicle dynamics with tire modeling by Pacejka magic formula was validated for example in the master thesis [14], where validation on a real car Porsche Boxster S was performed or in the master thesis [12], where validation on a student formula was performed. In addition, a similar model was validated on a student car in the author's previous work [22]. For more information about vehicle dynamics modeling, please look at the previously referenced [6], [13], [20]. Electric motor model and battery model are validated in the mentioned papers [10] and [3].

Chapter 4

Nonlinear Model Predictive Control

4.1 NMPC introduction

Model predictive control (MPC) is well-established advanced algorithm in control theory. Therefore, it will be described very briefly here. Up to date, extensive books dedicated to MPC are for example [2] and [18]. MPC is model-based algorithm. Dynamic design model (simplified) of the system is assumed to be derived. Mostly linear models for MPC are used, but also variants of MPC exist where this is not necessary and nonlinear models can be used - Nonlinear model predictive control (NMPC). In the thesis, NMPC is used. Generally, MPC algorithm can be simply described by three steps that are performed at **each discrete time** t

- get states $x(t)$ estimates based on measurements $y(t)$ of the system
- solve optimization problem on a given prediction horizon $(t, t + T_s, ..., t + NT_s)$ via inputs $\{u_1, u_2, u_N\}$ manipulation,
- select only the first input u_1 and apply it to the system

Discarding all inputs except the first one in the last step ensures that MPC is a control algorithm with feedback. In practice, a special form of the optimization problem is often used, where the cost function (the objective of the optimization) consists of matrices Q and R that weight states magnitudes

and control inputs magnitudes. However, other optimization formulations are possible in the second step (for instance, the optimization described later in the section 5.5.2).

The term NMPC will be used in the thesis for a tool provided by Garrett motion which implements the nonlinear MPC. NMPC will be briefly described on the following lines. Only parts needed for the thesis are covered in the summary. For further information see papers or literature on nonlinear model predictive control, for example, [2], [18]. One NMPC iteration overview is shown in algorithm 1. The NMPC works for continuous systems as well

Algorithm 1 Nonlinear model predictive control - one iteration overview.

1: **Inputs**: u_{init}, x_0
2: $u(t) \leftarrow u_{init}$
3: **while not** TermCond **do**
4: $\quad y_{sim}, x_{sim} \leftarrow \text{simulate}(u(t), x_0)$
5: $\quad H_y = \text{sensitivityCalculation}(u(t), x_{sim}, y_{sim})$
6: $\quad \delta u(t)* = \text{solveQP}(H_y, y_{sim}, u(t))$
7: $\quad u(t) \leftarrow u(t) + \alpha \delta u(t)*$
return $u(t)$

as for discrete systems, the version that uses continuous systems will be described and further used. Initial input u_{init} (eg. input from previous iteration) and current state x_0 of the system are inputs to the algorithm. Then while the termination condition is not met, iterations of sequential quadratic programming (SQP) are performed (line 3 - line 7). At the beginning of this SQP iteration, a system is simulated by an ordinary differential equation solver (eg. ode45) from the state x_0 by using piecewise constant input $u(t)$. It results in state and output trajectories x_{sim} and y_{sim}, but these are taken only at times given by sample time T_s. Sensitivity matrices H_y from perturbation δu of input $u(t)$ to the output y are computed at line 5. Jacobians of the state function $f(x, u)$ and output function $g(x, u)$ with respect to x and u along the trajectory x_{sim}, y_{sim} (at discrete times kT_s) are needed for this. Sensitivity H_y is defined as

$$\delta y(t) = H_y \delta u(t), \tag{4.1}$$

where $\delta y(t) = y_{pred} - y_{sim}$, thus

$$y_{pred} = y_{sim} + H_y \delta u(t). \tag{4.2}$$

NMPC tool cost function can be formulated (one of the possible formulations

including various terms) as

$$\begin{aligned} J_{out,in} = \frac{1}{2}[&(y_{pred} - y_{ref})^T Q_r (y_{pred} - y_{ref}) + y_{pred}^T Q y_{pred} + \\ &+ (u(t) + \delta u)^T R (u(t) + \delta u) + \Delta(u(t) + \delta u)^T R_d \Delta(u(t) + \delta u) = \\ &= \frac{1}{2}[(y_{sim} + H_y \delta u(t) - y_{ref})^T Q_r (y_{sim} + H_y \delta u(t) - y_{ref}) + \\ &+ (y_{sim} + H_y \delta u(t))^T Q (y_{sim} + H_y \delta u(t)) + \\ &+ (u(t) + \delta u)^T R (u(t) + \delta u) + \Delta(u(t) + \delta u)^T R_d \Delta(u(t) + \delta u)], \end{aligned} \tag{4.3}$$

where the first term belongs to reference tracking cost, the second term to the output magnitude cost, the third term to the input magnitude cost and the last term to the cost related to a change between consecutive inputs (Δ denotes to differences between consecutive inputs).

Constraints y_{min}, y_{max} on outputs are formulated as a soft constraint. Therefore, these constraints are transformed to the cost function as

$$J_{soft} = \frac{1}{2} e^T G e, \tag{4.4}$$

where the new variable e ensures this (see formulation 4.6 below). General cost function can be written down as

$$J(\delta u, e) = J_{out,in} + J_{soft}. \tag{4.5}$$

And the optimization problem is formulated as follows

$$\begin{aligned} \min \quad & J(\delta u, e), \\ \text{subject to} \quad & u_{min} \leq u(t) + \delta u(t) \leq u_{max}, \\ & y_{min} + e \leq y_{sim} + H_y \delta u(t) \leq y_{max} + e, \\ & e \geq 0. \end{aligned} \tag{4.6}$$

This quadratic program is solved. The resulting optimal $\delta u(t)*$ is multiplied by step size parameter $\alpha \in (0, 1\rangle$ and added to the previous input (line 7). SQP iterations are ended when either the number of iterations is exceeded or if the difference between the previous and the current input is less than certain constant (parameter).

4.2 NMPC vehicle design models

In this section, design models (in some literature the term control oriented model is used) of the vehicle for NMPC algorithm are introduced at different

levels of complexity. At the end of the section, a comparison of design models with high fidelity model is done.

After a closer look to the NMPC algorithm introduction, the model used for optimization has to be as simple as possible. Especially, the sensitivity calculation part is restrictive. Mainly the convexity of a model is required for the correct function of the algorithm. Convexity is required to achieve global optimality, not only local. Linearity is not necessary as nonlinear MPC is used. However, it is advantageous as it improves convergence.

Two models for NMPC algorithm in slightly different complexity levels are introduced. Functions of the high fidelity model that can cause problems were replaced by appropriate replacements. Further aim was to reduce the number of states while maintaining the system behavior. For these purposes, we also took into account the sample time then used for NMPC.

4.2.1 High complexity vehicle design model

We will go through the individual components, derive the vehicle design model equations and comment changes or simplifications.

Electric motor

Both states of the electric motor were neglected. These states are not necessary. The first state T_{int} (Eq. 3.1) can be neglected because it tracks T_{req} and its dynamics is very fast compared with the sample time intended for NMPC algorithm (0.1s, the model sample time is discussed later in the section 4.4). The second state ω_{em} (Eq. 3.2) is neglected as well. The dynamics given by the time constant t_ω is fast too, but the main reason for doing this is that the moment of inertia of the electric motor is very small compared with other moments of inertia on which the output torque acts. Therefore, the torque "taken" by EM for its angular acceleration is negligible. Remaining relevant equations are

$$T_{em} = T_{req} r_\omega, \tag{4.7}$$

$$P_{em} = T_{req} r_\omega \omega_{in} \eta_{-1}(T_{req}). \tag{4.8}$$

The result of this simplification in comparison with the original system can be seen in the figure 4.1.

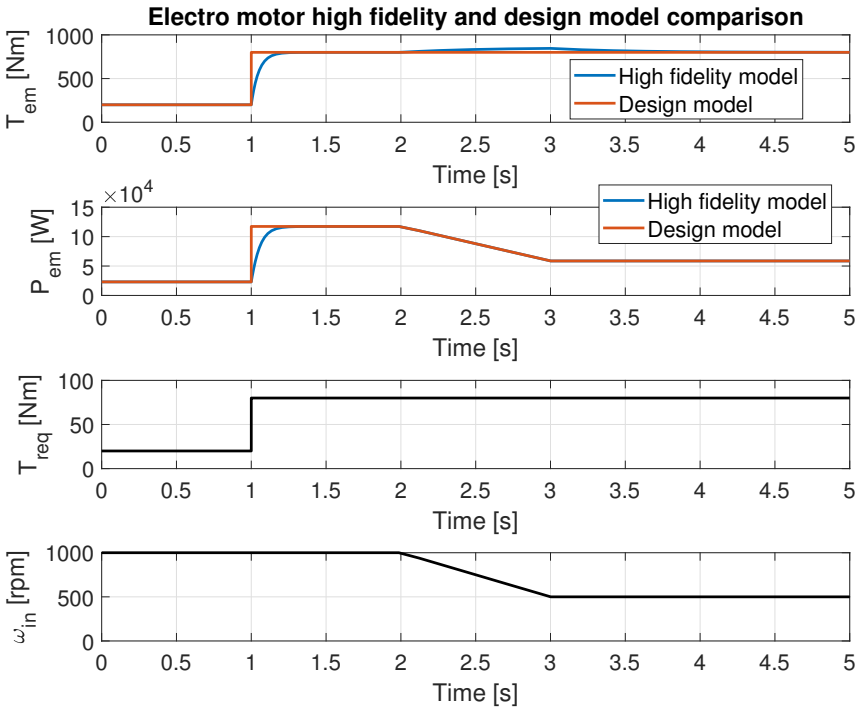


Figure 4.1: Validation of electric motor simplifications.

Battery

For the battery model, it is important to get rid of the square root in the equation 3.10. The square root makes trouble when the number in it is negative. NMPC algorithm has all output constraints as soft constraints, therefore the non-negativity cannot be guaranteed by this constraint and the algorithm could fail. Simplification of the battery model was done with the following assumptions. Transient events caused by capacitors are neglected, therefore RC pairs can be discarded and one resistor as $R = R_0 + R_1 + R_2$ is introduced. When a comparison of the open circuit voltage V_{oc} with voltage lost at this resistor (Ri) is done it can be find out that even for big values of current i, $V_{trm} \approx V_{oc}$ holds in the order of units of voltages. Then the equations describing this reduced battery model are

$$\dot{SoC} = -\frac{100}{3600C} i, \tag{4.9}$$

$$i = \frac{P_{trm}}{V_{oc}}, \tag{4.10}$$

where $V_{oc} = f_{ocv}(SoC)$ or can be approximated eg. by a linear function in the relevant operating point (because SoC is changing very slowly within the

3s prediction horizon intended for NMPC - see 4.4). Justification or rather validation of these simplifications can be seen in the figure 4.2, where original and reduced models are compared.

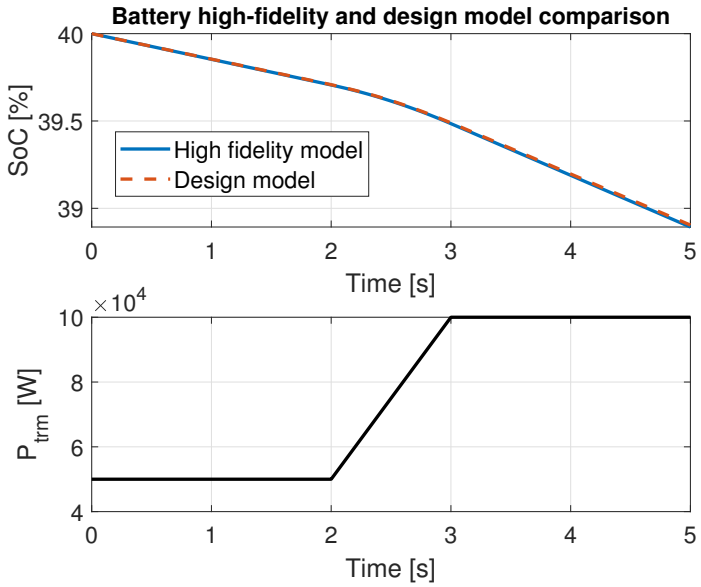


Figure 4.2: Validation of battery simplifications.

Wheel model

For the reduced wheel model, the aim is to get rid of equation 3.15 for λ together with equation 3.21 for wheel angular speed dynamics. There are two reasons for doing that. First, the former equation runs into error when the denominator is close to zero (both ω and v are close to zero), the latter equation dynamics is very fast and can cause stiff problems. If zero slip is assumed and thus tire peripheral velocity equal to vehicle speed, these equations can be replaced by the simpler one (here only forward motion is assumed (because of $T_{\omega,br}$))

$$F_{wx} = \frac{(T_\omega - T_{\omega,br})}{R}. \tag{4.11}$$

Furthermore, in Eq. 3.16 prevention of division by zero is not guaranteed. In the high-fidelity model implementation, $atan2(x_2, x_1)$ function is used for this purpose. This function is not differentiable at point (0, 0) and its usage is therefore unwise in a control oriented model for NMPC, which relies on it. This $atan2()$ function was replaced by the function

$$\hat{atan2}(v_{\omega y}, v_{\omega x}) = \frac{4v_{\omega y}}{\sqrt{1 + 3(4v_{\omega x})^2 + (4v_{\omega y})^2}}. \tag{4.12}$$

The function 4.12 was designed to accurately capture values for $v_{\omega x} > 1$. There are depicted function values for different $v_{\omega x}$ cuts in figure 4.3.

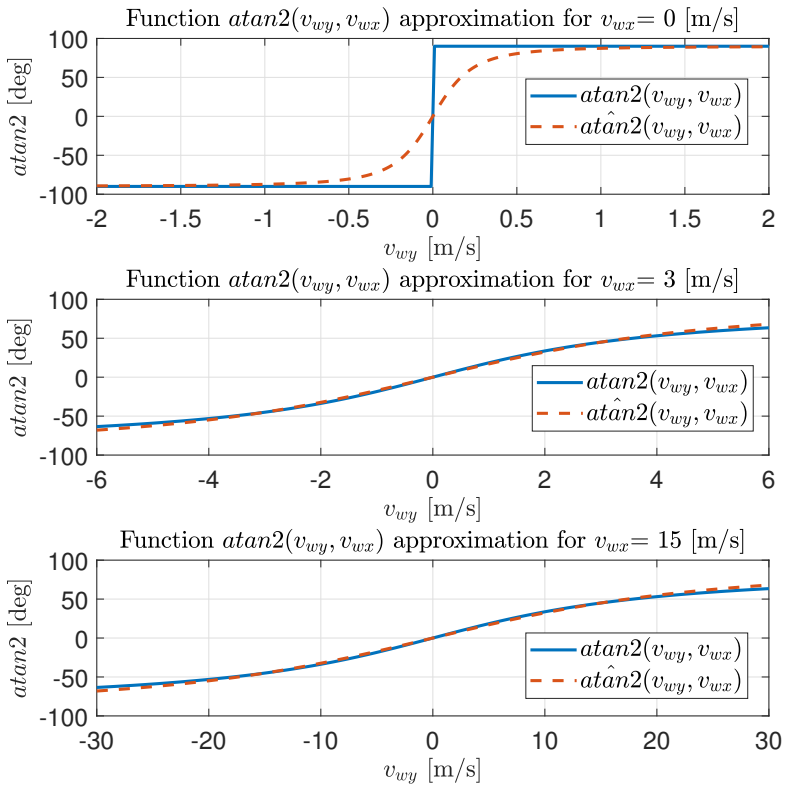


Figure 4.3: Validation of atan2 function simplification.

Derived reduced model is defined as follows

$$v_{wx} = v_x \cos\delta + v_y \sin\delta, \tag{4.13}$$
$$v_{wy} = -v_x \sin\delta + v_y \cos\delta, \tag{4.14}$$
$$\alpha = -\hat{atan2}(v_{\omega y}, v_{\omega x}), \tag{4.15}$$
$$F_{wx} = \frac{(T_\omega - T_{\omega,br})}{R}, \tag{4.16}$$
$$F_{wy}(\alpha) = F_z D_y \sin(C_y \arctan(B_y\alpha - E_y(B_y\alpha - \arctan(B_y\alpha)))), \tag{4.17}$$
$$F_x = F_{wx}\cos\delta - F_{wy}\sin\delta, \tag{4.18}$$
$$F_y = F_{wx}\sin\delta + F_{wy}\cos\delta. \tag{4.19}$$

Note that Eq. 4.17 is nonconvex (see Fig. 3.5b), but this can be eliminated by appropriate α restriction in NMPC optimization objective because roughly $\alpha \in (-10; 10)$ deg is desired.

Vehicle dynamics

Finally, the vehicle dynamics model was reduced. The only problem is with the side slip angle β dynamics (Eq. 3.28), when vehicle velocity is close to zero. There are two possible solutions. The first one is the replacement of the velocity magnitude v and its direction given by β with two velocities - one velocity in the vehicle x axis direction and one velocity in the y axis direction - this could be also done in the high-fidelity model. The other solution is to omit the side slip angle β. This is quite reasonable (for a control oriented model) because its values are relatively small during the ride. The second approach was implemented and β was neglected. Derived equations are then

$$F_x = F_{xf} + F_{xr} - F_{aero} - F_{slope}, \tag{4.20}$$
$$M_z = l_f F_{yf} - l_r F_{yr}, \tag{4.21}$$
$$\dot{e} = v\cos(\psi), \tag{4.22}$$
$$\dot{n} = v\sin(\psi), \tag{4.23}$$
$$\dot{\psi} = \dot{\psi}, \tag{4.24}$$
$$\dot{v} = \frac{F_x}{m}, \tag{4.25}$$
$$\ddot{\psi} = \frac{1}{I_z} M_z, \tag{4.26}$$
$$v_{xf} = v, \tag{4.27}$$
$$v_{xr} = v, \tag{4.28}$$
$$v_{yf} = l_f \dot{\psi}, \tag{4.29}$$
$$v_{yr} = -l_r \dot{\psi}. \tag{4.30}$$

Validation

Vehicle design model was created with 6 states - e, n, ψ, v, $\dot{\psi}$ and SoC - instead of 15 states for the original model. Comparison of these integrated models is shown in the figure 4.4. Three states - east and north position and velocity - depending on the three inputs to the front wheel are shown in the figure. State of charge has been already validated and heading ψ and possibly $\dot{\psi}$ can be deducted from the first subfigure. Moderate differences in the velocity course can be mainly attributed to the slip ratio neglecting. Therefore, the design model acceleration is higher than the one of high fidelity model. Differences in position are caused by velocities and by the side slip angle β neglecting. However, these differences have arisen on the horizon of 10 seconds. The time horizon intended for NMPC algorithm is much smaller

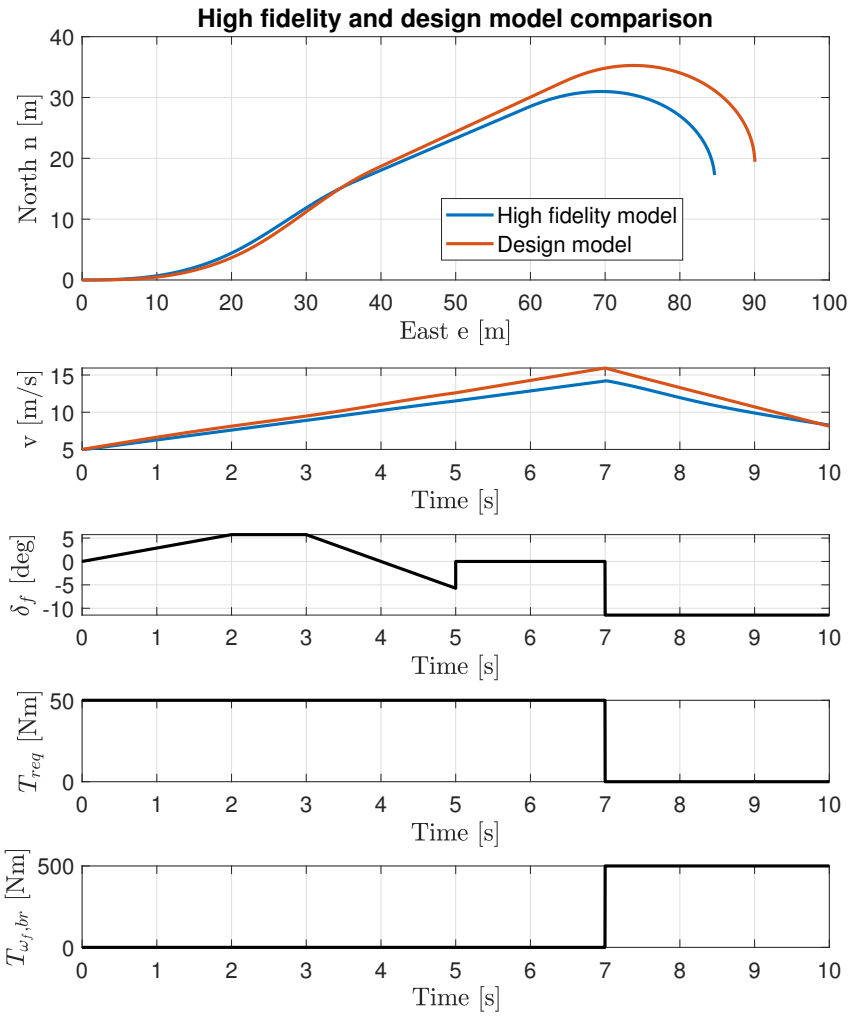


Figure 4.4: Validation of the high complexity vehicle design model.

(3 seconds). Thus, this is an acceptable mismatch for the control oriented model.

4.2.2 Low complexity vehicle design model

For this complexity level, the derived design model in the previous subsection is further simplified. The main simplification besides the previous model is in the wheel component. Then state $\dot{\psi}$ will be neglected. All remaining

functions and states are maintained - vehicle position, yaw, velocity and state of charge (consumption minimization purpose).

Electric motor and battery

Equations remain the same as in the high complexity design model 4.2.1 (nothing to reasonably further simplify),

$$T_{em} = T_{req} r_\omega, \tag{4.31}$$
$$P_{em} = T_{req} r_\omega \omega_{in} \eta_{-1}(T_{req}), \tag{4.32}$$
$$\dot{SoC} = -\frac{100}{3600C} i, \tag{4.33}$$
$$i = \frac{P_{trm}}{V_{oc}}. \tag{4.34}$$

Wheels and vehicle dynamics

The lateral forces generation by using Pacejka magic formula was neglected. Turning is performed directly by the front wheel steering. The remaining equation for both front and rear wheels is

$$F_x = \frac{(T_\omega - T_{\omega,br})}{R}. \tag{4.35}$$

Vehicle dynamics without neglected yaw-rate state is as follows

$$F_x = F_{xf} + F_{xr} - F_{aero} - F_{slope}, \tag{4.36}$$

$$\dot{e} = v\cos(\psi), \tag{4.37}$$
$$\dot{n} = v\sin(\psi), \tag{4.38}$$
$$\dot{\psi} = \frac{1}{k} v \delta_f, \tag{4.39}$$
$$\dot{v} = \frac{F_x}{m}, \tag{4.40}$$

where k is a constant that best fits to the given vehicle. The constant $k = 3.125\,\mathrm{m}$ is used in this work. The simplification introduced in Eq. 4.39 is based on a mathematical analysis of the physical behavior. The constant k can be interpreted as a turning radius.

Validation

A control oriented model with 5 states - e, n, ψ, v and SoC was derived. Comparison of the design model and of the high-fidelity model is shown in the figure 4.5. Performance of the design model is reasonable (for the purpose of control in time horizon within a few seconds) while having much simpler structure.

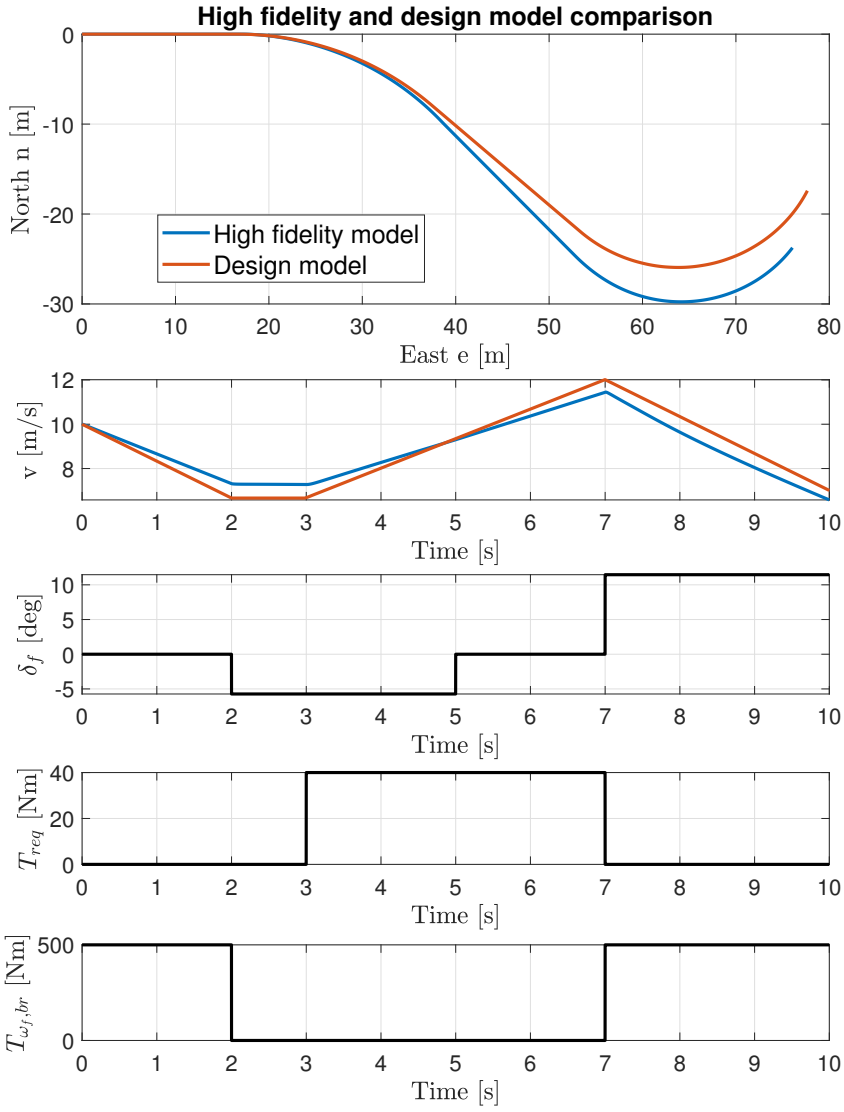


Figure 4.5: Validation of the low complexity vehicle design model.

4.3 NMPC problem solution

In this section, the problem solution by NMPC is described. First, different alternative approaches are shown and discussed. Then the used solution is explained. Presence of static obstacles and more challenging moving obstacles in the environment is the main problem for NMPC algorithm. Obstacles generally create a nonconvex space, which is to be dealt with. Within state of the art research of MPC in a nonconvex environment, two approaches are often used. The first one is by using suitable heuristics. Heuristics give a path or constraints for the path and then MPC is used to ensure that the trajectory is within the desired path or directly track the path. For example, articles [11], [8], [7] use this approach. Mixed-integer programming [19] is alternative approach that is computationally very demanding.

4.3.1 Mixed-integer programming

Mixed-integer programming (MIP) is used in applications with both integer and real valued variables. For example mixed-integer linear program can be formulated as

$$\begin{aligned} \min \quad & \mathbf{c_1^T z} + \mathbf{c_2^T x}, \\ \text{subject to} \quad & \mathbf{A_1 z} + \mathbf{A_2 x} \leq \mathbf{b}, \\ & \begin{bmatrix} \mathbf{z} \\ \mathbf{x} \end{bmatrix} \geq \mathbf{0}, \\ & \mathbf{z} \in \mathbb{Z}^{n_1}, \mathbf{x} \in \mathbb{R}^{n_2}. \end{aligned} \tag{4.41}$$

Similarly, a mixed integer program that is quadratic in real variables can be formulated. Dedicated solvers to these problems are available. Within control problems, eg. YALMIP can handle MIP. The MIP problems are often solved via

- enumeration method,
- branch and bound method,
- cutting planes method.

For example, following requirement 4.42 shown in figure 2.2,

$$\begin{aligned} &\textbf{if } (\mathit{east} \geq 60 \textbf{ and } \mathit{east} \leq 70) \textbf{ then} \\ &\quad \mathit{north} \geq 20 \textbf{ xor } \mathit{north} \leq 2. \end{aligned} \tag{4.42}$$

can be expressed with two new binary variables $z_1, z_2 \in \{0; 1\}$ as 4.43 and 4.44,

$$\begin{aligned} &\textbf{if } (east \geq 60 \textbf{ and } east \leq 70) \textbf{ then} \\ &z_1(north - 20) + z_2(2 - north) \geq 0, \qquad (4.43) \\ &z_1 + z_2 = 1. \qquad (4.44) \end{aligned}$$

Similarly, other logical statements can be expressed with binary variables. The solution by so-called big M (big positive number) is often used when the first statement or the second statement must hold. However, despite the easy program building mixed-integer programs are NP-hard and its usage is thus very limited.

The thesis uses NMPC tool by Garrett motion, where MIP is not available. To avoid binary variables, the equations can be modified with one additional variable $z \in \mathbb{R}$ and one additional soft constraint y_{soft} as

$$\begin{aligned} &\textbf{if } (east \geq 60 \textbf{ and } east \leq 70) \textbf{ then} \\ &z(north - 20) + (1 - z)(2 - north) \geq 0, \qquad (4.45) \\ &y_{soft} = z^2 - z = 0. \qquad (4.46) \end{aligned}$$

Although this formulation uses only real variables, it cannot be used for NMPC because the problem is nonconvex and therefore stuck at local minima (with z near 0 or with z near 1) is very likely. Due to that fact, the approach cannot be applied.

4.3.2 Dealing with static obstacles - Cost to go heuristics

For dealing with static obstacles, a solution based on heuristics called Cost to go [19] was chosen. Points in the space which are beneficial to go through compared to the other points are the result of the heuristics. These points are given in advance, it means before running the NMPC. However, the overall algorithm was proposed such that NMPC still generates the trajectory, but only in given restricted part of the space. The heuristics takes points in the space and assigns a value to each of those points, such that its value measures cost of the trajectory from that point to the goal. The points were chosen as vertices of the obstacles. The cost can be based on various optimization objectives.

The proposed solution is expressed in algorithm 2. Overall approach in every iteration is: (1) based on Cost to go heuristics restrict the possible

space, (2) get the optimal inputs by NMPC, (3) provide the first optimal input.

Algorithm 2 Nonlinear model predictive control with **Cost to go** heuristics.

1: **Inputs**: x_0, obstacle vertices V, obstacles O
2: $G = \text{createGraph}(O, V, goal)$
3: $T = \text{Dijkstra}(G)$
4: $i \leftarrow 0$
5: **while** $i \leq n$ **do**
6: $\quad e_0, n_0 \leftarrow x_0(1), x_0(2)$
7: $\quad v, path \leftarrow \text{findBestNode}(T, O, e_0, n_0)$
8: $\quad y_{softLim} = \text{restrictTrajectory}(path, O, V)$
9: $\quad y_{ref} = \text{setRefAndWeights}(path, x_0)$
10: $\quad x_{0c} \leftarrow x_0(\text{idxs})$
11: $\quad u = \text{NMPC}(x_{0c}, y_{ref}, y_{softLim})$
12: $\quad x_0 \leftarrow \text{simulateHF}(x_0, u(1), T_s)$
13: $\quad i \leftarrow i + 1$

Individual steps of the algorithm will be described on the following lines. The second test scenario will serve for illustration.

Inputs are the initial state x_0 of the system, static obstacles O - convex polygons and the obstacle vertices V. It is always possible to split nonconvex polygons to convex polygons. Graph is created from the vertices at line 2 (example shown in Fig. 4.6). Its nodes are obstacle vertices and goal. Edges are collision-free connections between the vertices (lines without collision with obstacles). Obstacle edges are considered as collision free. The common line is not collision free in the case where the obstacle is a result of nonconvex polygon split.

Next part of the algorithm (line 3) assign each of the vertices of the graph G cost based on Dijkstra search. The cost can be based on a combination of optimization parameters $J = J(t, SoC, length)$ and computed by suitable optimization tool. Cost based on the length $J = J(length)$ is used in the work. The search starts from the goal. At the end, each vertex has a cost and its parent. Therefore, Dijkstra search creates a tree T from the graph with the goal node as a root (fig. 4.7).

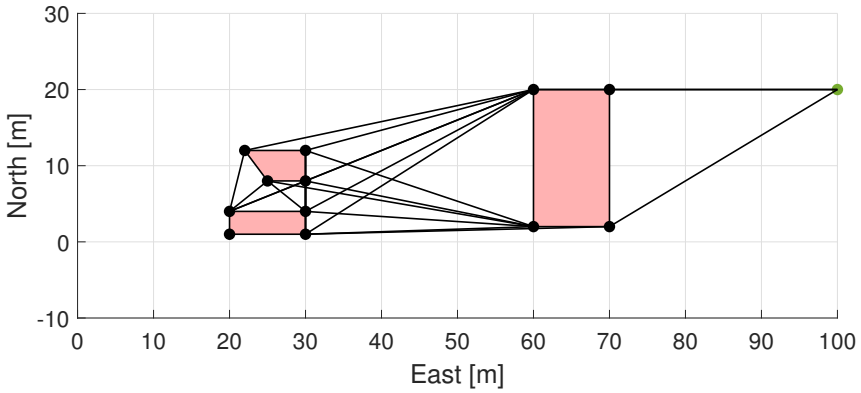


Figure 4.6: Created graph: nodes - obstacles vertices and goal, edges - collision free lines between vertices

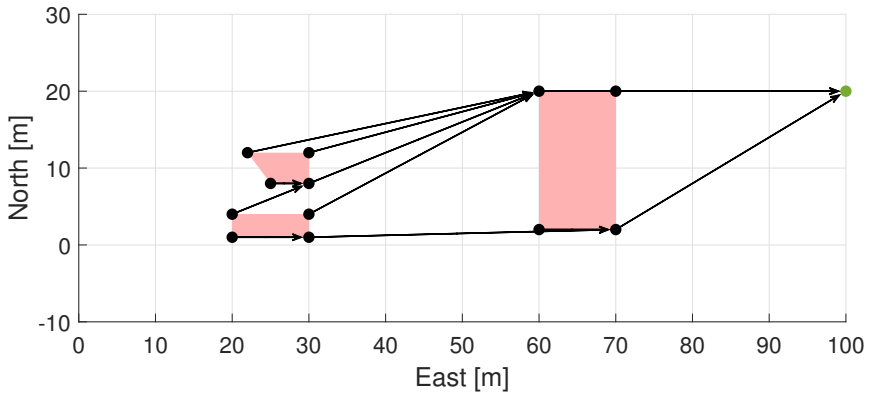


Figure 4.7: Tree created by Dijkstra's algorithm from graph. Goal as a root, vertices as nodes.

Then the while loop runs while the number of iterations given by the simulation time is not exceeded. Note that graph creation and Dijkstra search can be moved into the loop if needed. For example, if only some obstacles are known at the start and knowledge of the environment develops. The vehicle current position can be included in the graph in that case. If not (as in the presented algorithm), the node u which gives best cost for the current vehicle position $J((e,n))$ is chosen. The cost used in this work is

$$J((e,n)) = J(u) + J((e,n),u) = J(u) + length((e,n),u). \tag{4.47}$$

At the same time, the node has to have collision-free line connection with vehicle. The node u also gives the optimal path (based on the heuristics) to the goal (Fig. 4.8). The function J has two meanings: $J(\cdot)$ means node

cost, i.e. cost from node to root, whereas $J(\cdot,\cdot)$ means cost from one node to another.

The heuristics path serves only for trajectory restrictions because of obstacles. The trajectory restriction by lines is shown in the figure 4.9. Obstacles along the trajectory are delimited by lines. Only nodes visible from the vehicle position can be delimited and only obstacles on some distance horizon are delimited. These requirements are expressed by outputs soft limits $y_{softLim}$.

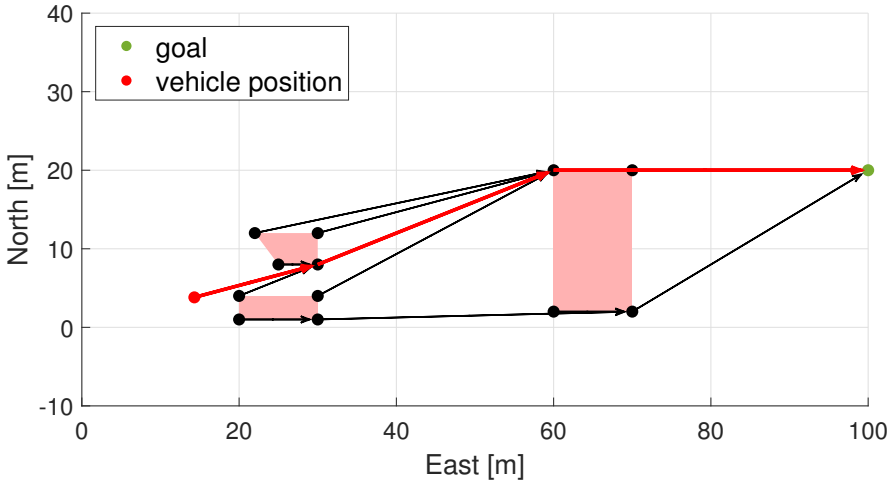


Figure 4.8: The best path given by vehicle position.

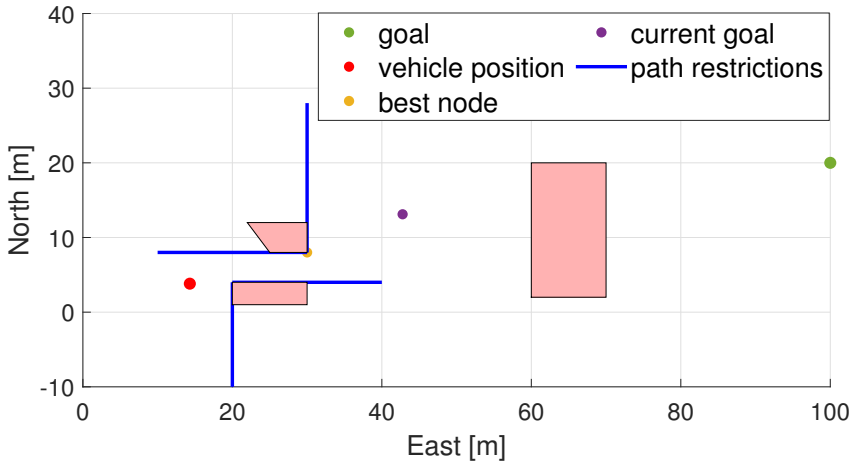


Figure 4.9: Trajectory restricted by lines on some distance horizon.

References and their weights are set at line 9. Such a point on *path* is chosen

as a current goal that the vehicle is not able to reach it within one prediction horizon. More precisely, the point (current goal) is reachable by the vehicle if the vehicle goes exactly along the path, uses the maximal acceleration from current velocity while respecting the maximal allowed velocity. The choice of the current goal is intentional, because then the vehicle does not brake unnecessary at intermediate goals and goes the right path. If some vertices are on the path between the vehicle and the current goal, the position of the vertices is set as a reference in the middle of the prediction horizon. With the similar procedure as for the current goal, the vertex is set as a reference for a particular time in the prediction horizon. These points serve as guiding rather than actual references. Therefore, weight on the reference points is set as linearly increasing with the prediction time. Thus, the biggest weight is assigned to the current goal and weights of points near the vehicle approach zero.

However, the situation is slightly different if the part of the Dijkstra path from vehicle position to the current goal has a sharp turn. The situation is outlined in the figure 4.10. For this case, the presented algorithm would quite easily stuck at local minima behind the obstacle as the cost is not improving along the obstacle edge towards the nearest vertex. As described above, the setting vertices at the path as other references at particular times in the prediction horizon helps, however, it can be insufficient in some cases.

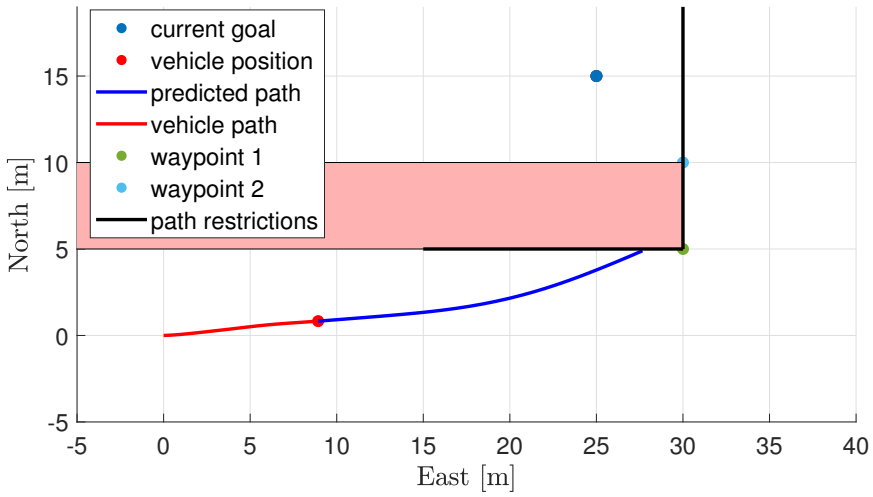


Figure 4.10: Dijkstra path positions directly as references - stuck at local minima is possible.

Better solution for the cases with sharp turn path is shown in the figure 4.11. In this case, only nearest vertex on the best path is set as a reference for time in the prediction horizon as described above. It is also possible to set as a reference position given by obstacle edge length ("current goal" in the

figure) as shown in the figure. In this case (sharp path), the nearest vertex on the best path has bigger weight because the following position is only auxiliary.

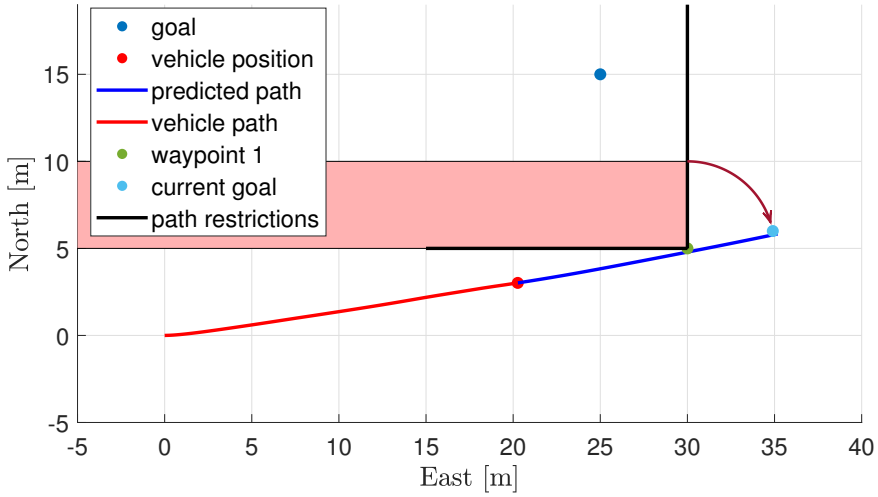


Figure 4.11: Modification for paths with sharp turn.

Only states needed for design model x_{0c} are chosen from vehicle states x_0. Then NMPC algorithm is run. The first optimal input is applied for sample time T_s and the high-fidelity model (or real system if available) is simulated.

To summarize this part, the procedure is following: the heuristics select a subspace of the state space. Then NMPC generates a trajectory in this subspace. The NMPC actually generates the trajectory compared to conventional trajectory tracking problem.

4.3.3 Dealing with moving obstacles

In the previous subsection, dealing with static obstacles was solved by heuristics Cost to go. It is not possible to use the described approach also for moving obstacles. The reason is the position change in time. Knowledge of the moving obstacle positions on the prediction horizon is considered. Vertices of an obstacle (polygon) are moving. Therefore, it is not possible to assign costs to vertices as in the previous section. Assignment of each vertex cost for individual time steps in the prediction horizon is realizable. The used heuristics in the previous section decides about the side to go around an obstacle. In this case, the decision to go around the obstacle from the selected

side can be often unfeasible from the vehicle dynamics constraints.

Instead, the moving obstacle is wrapped by some function, circumscribed by eg. ellipse (fig. 4.12). Then an output soft constraint is created. The constraint ensures that positions of the controlled vehicle in the prediction horizon are outside the ellipse.

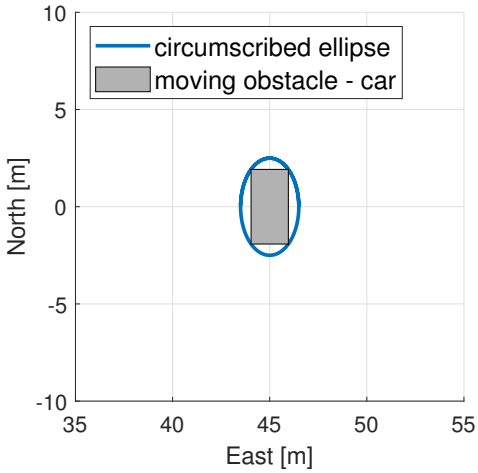


Figure 4.12: Dealing with moving obstacles.

It is important to say that this approach can lead to stuck at local minima. Hence, it is the reason why this procedure is not applied to static obstacles. This is a subject for further improvements in the future.

4.4 NMPC results

In this chapter, results of NMPC method are presented. Outcomes are given and described for scenarios (Fig. 2.1), (Fig. 2.2) and (Fig. 2.3) by NMPC algorithm. For all scenarios, a zero inclination angle $\alpha_{inc} = 0$ is considered. Code was implemented in MATLAB with NMPC tool provided by Garrett motion. The NMPC runs on design (control oriented) models, the simulation on high fidelity model.

The NMPC algorithm parameters were chosen as

- model sample time: $T_s = 0.1s$,

- prediction horizon samples: $N = 30$,
- $\implies$ prediction horizon time: $T = NT_s = 3s$.

Option of these parameters provides a suitable combination of model sample time and prediction horizon time. Model sample time is short enough to capture majority of the vehicle dynamics. However, some modes of the dynamics are too quick (for instance, slip ratio) and it cannot be properly controlled with this model sample time, but anyway it is not the thesis objective. Number of prediction samples gives three-second long prediction time. The prediction time is very short. Searching for the best trajectory in a chosen subspace for minimization of time or battery state of charge includes only 3s prediction. Or, for instance, it can be impossible to stop from high speed, especially when slip is present. However, longer prediction time as well as shorter model sample time means higher time demands for computations and more complex space to inspect. The 3 seconds long prediction horizon is tradeoff respecting the aforementioned facts.

Formulation of time minimization is not straightforward for the classic model predictive control. Prediction horizon is fixed and it is not an optimization parameter. Thus, it is not possible to set a goal (current goal) position as an end state and minimize the time to reach that goal, while satisfying other constraints. Instead, the current goal and waypoints are set as an reference for certain sample(s) in the prediction horizon. Therefore, the formulation is the minimization of distance to the goal rather than time minimization. The procedure of current goal and way-point setting as a reference for prediction samples is described in section 4.3.2. The weight of the main sample (sample with the biggest weight on it, mostly the current goal) in the prediction horizon for position tracking will be denoted as w.

Only inputs to the front axle are controlled, therefore the subscript f denoted to the front axle is redundant. The inputs are bounded by values

- $\delta \in \langle -25\,\mathrm{deg}; 25\,\mathrm{deg} \rangle$,
- $T_{req} \in \langle -100\,\mathrm{N\,m}; 100\,\mathrm{N\,m} \rangle$ ($drive\ torque \in \langle -1000\,\mathrm{N\,m}; 1000\,\mathrm{N\,m} \rangle$),
- $T_{\omega,br} \in \langle 0\,\mathrm{N\,m}; 2000\,\mathrm{N\,m} \rangle$,

for all scenarios. Note that T_{req} is requested torque for an electric motor, while $T_{\omega,br}$ is brake torque acting directly on the wheel. There is a gear ratio $r_\omega = 10$, which transforms out torque of the electric motor generator

to a torque acting on the wheel. In the following figures, the expression *drive torque* is denoted to the term $T_{req}r_{\omega}$ (to have the same scale as for brake torque). The matrix R is not used in the cost function, the matrix R_d that weights the differences between consecutive inputs is preferred as this option represents better solved tracking reference problem. Therefore, in the scenarios NMPC cost function J_{in}, representing the cost for inputs, is given by (roughly, may slightly differ depending on the scenario) matrices

- $R_{d_\delta} = diag(50), \qquad R_{d_{T_{req}}} = diag(10), \qquad R_{d_{T_{\omega,br}}} = diag(10),$

which weights inputs that are normalized in interval $\langle -1; 1\rangle$ in advance for this purpose. The matrices are square of size 30x30 with stated values on their diagonals. Requirements (constraints) for the individual scenarios will be formulated by linear temporal logic. Section 5.1.1 is denoted to linear temporal logic, but the requirements are very simple in the presented cases. Low-complexity design model is used in the scenario 1 and 3, high-complexity design model is used in the scenario 2.

4.4.1 Scenario 1

The first scenario is without any obstacles. Control of the vehicle will be shown for two different costs on the state of charge maximization. The NMPC cost is as

$$\begin{aligned} J_{out,in} = \frac{1}{2}[&(e_{pred} - e_{ref})^T Q_{r_e}(e_{pred} - e_{ref}) + \\ &+ (n_{pred} - n_{ref})^T Q_{r_n}(n_{pred} - n_{ref}) + \\ &+ (SoC_{pred} - SoC_{ref})^T Q_{r_{SoC}}(SoC_{pred} - SoC_{ref})] + \\ &+ J_{in}, \end{aligned} \tag{4.48}$$

where matrices Q_{r_e} and Q_{r_n} weight position reference tracking and $Q_{r_{SoC}}$ state of charge tracking. Here, the environment is without obstacles, therefore according to the proposed algorithm, only the current goal is given as a reference for the last sample in the prediction horizon and weight for it was chosen as $w = 0.1$ (last element on diagonal of both matrices Q_{r_e}, Q_{r_n}). Slightly bigger SoC then current SoC is given as a reference for state of charge: $SoC_{ref} = SoC + 2\%$ at the end of the prediction horizon. Paths (Fig. 4.13) and other states and inputs (Fig. 4.14) of the vehicle are shown in the following figures for two different weights on SoC. The first one shows the situation without cost for SoC, the other one with cost on SoC given by the square matrix $Q_{r_{SoC}} = diag(0, .., 0, 10)$.

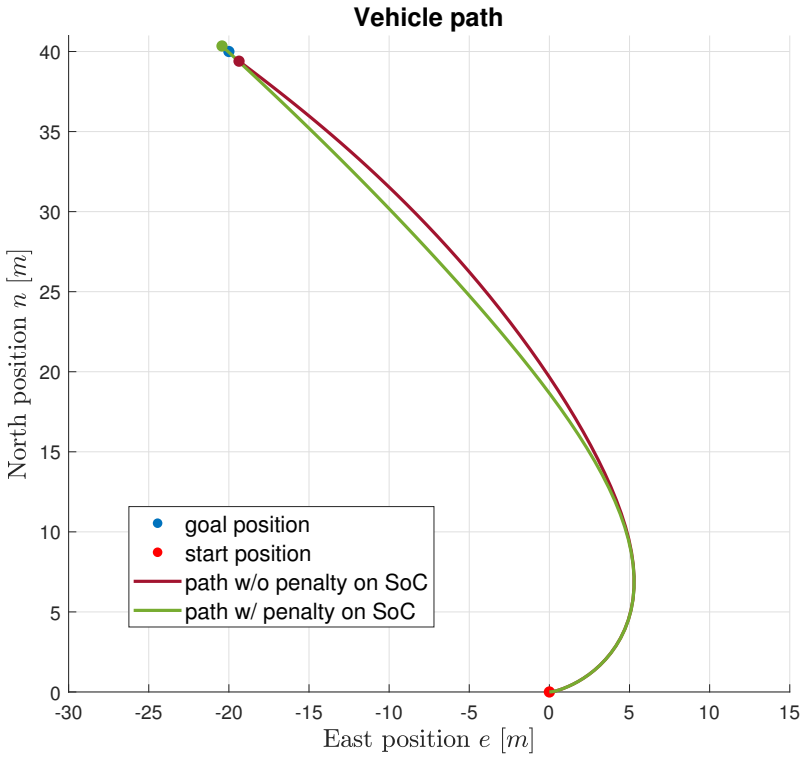


Figure 4.13: Vehicle path by NMPC in the first scenario. Two versions of the cost function compared - with(w/) and without (w/o) penalty in the cost function for state of charge maximization.

The path with penalty on the state of charge maximization is more direct than the other path. At the final positions of shown paths, the vehicle still has a certain (small) velocity. It can be seen that the presence of the penalty for SoC in the cost function can inconveniently affect tracking of the goal - it is advantageous (from the optimization point of view) to slightly pass the goal, because long-lasting mild braking gives better recuperation power than quick heavy braking.

Other initial states of the vehicle are $v_{init} = 2\,\frac{\mathrm{m}}{\mathrm{s}}$, $\psi_{init} = 0\,\mathrm{deg}$, $SoC_{init} = 40\,\%$. From the subfigure which displays SoC (Fig. 4.14) can be seen that trajectory with penalty on state of charge maximization ends with value about $SoC = 39.8\,\%$, while the other trajectory ends with value only slightly better than $SoC = 39.7\,\%$. The reason is shown in the last subfigure. For the trajectory with consumption minimization, braking by motor generator torque is preferred over brake torque. A smooth course of the steering angle δ is shown for both trajectories.

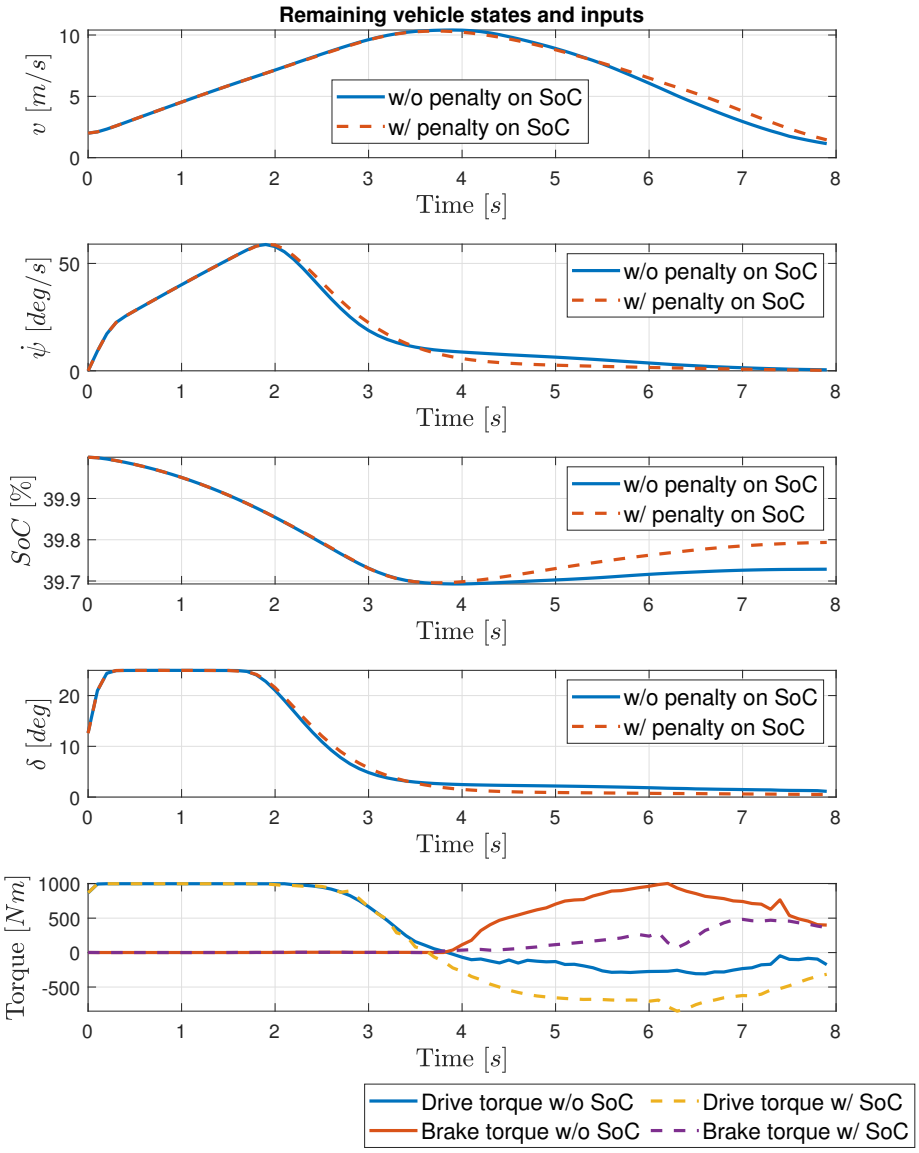


Figure 4.14: Other vehicle states and inputs in the first scenario. Two versions of cost function compared - with(w/) and without (w/o) penalty in the cost function for state of charge maximization.

4.4.2 Scenario 2

Static and moving obstacles are present in the second scenario. The NMPC tracking cost formulation is as in the previous scenario 4.48. The matrices Q_{r_e} and Q_{r_n}, which weights position reference tracking, are set during the

ride according to the proposed algorithm. Weight for the main sample (main position reference) was chosen as $w = 0.2$.

The obstacles avoidance requirements can be formulated by linear temporal logic as

$$\Box\neg(\text{collision with static obstacle}), \tag{4.49}$$
$$\Box\neg(\text{collision with moving obstacle}), \tag{4.50}$$

where the collision means that the vehicle position is in the obstacle. The requirements are reformulated to MPC constraints by the described proposed algorithm with the usage of soft output constraints for obstacles restriction.

Two NMPC cost functions for different penalties for SoC are compared in this scenario, too. The first cost is given by matrix $Q_{r_{SoC}} = diag(0, .., 0, 30)$, the other one by $Q_{r_{SoC}} = diag(0, .., 0, 100)$. Paths (Fig. 4.15) and other states and inputs (Fig. 4.16) of the vehicle are shown in the figures. The vehicle is quicker in the situation with lower SoC penalty and it chooses an overtaking maneuver for moving obstacle avoidance, while for bigger SoC penalty it maintains more economy trajectory and passes the moving obstacle from the other side. For a better understanding of the described situation, look at the predictions for certain simulation times shown in the figure 4.17 for both cases with the depicted moving obstacle.

Other initial states of the vehicle are $v_{init} = 0.1\,\frac{\text{m}}{\text{s}}$, $\psi_{init} = 0\,\text{deg}$, $SoC_{init} = 40\,\%$.

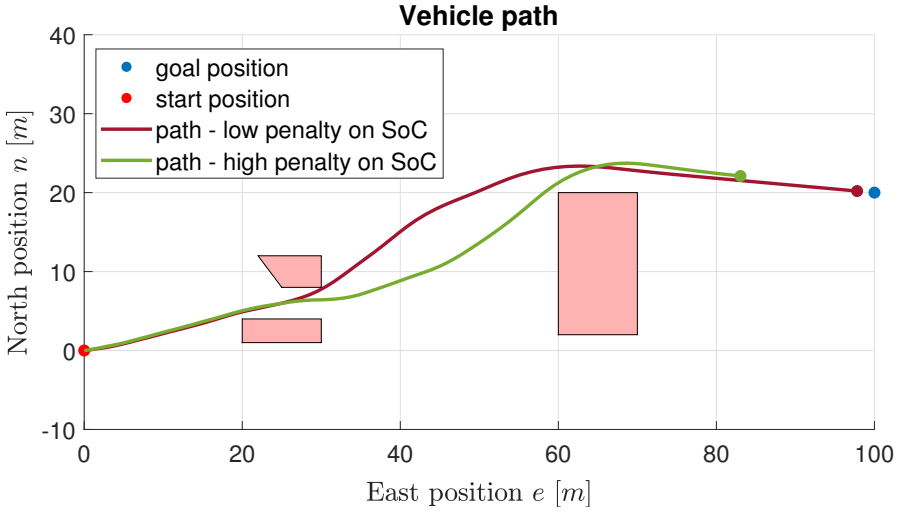


Figure 4.15: Vehicle path by NMPC in the second scenario. Two versions of cost function compared - with low and with high penalty in the cost function for the state of charge maximization.

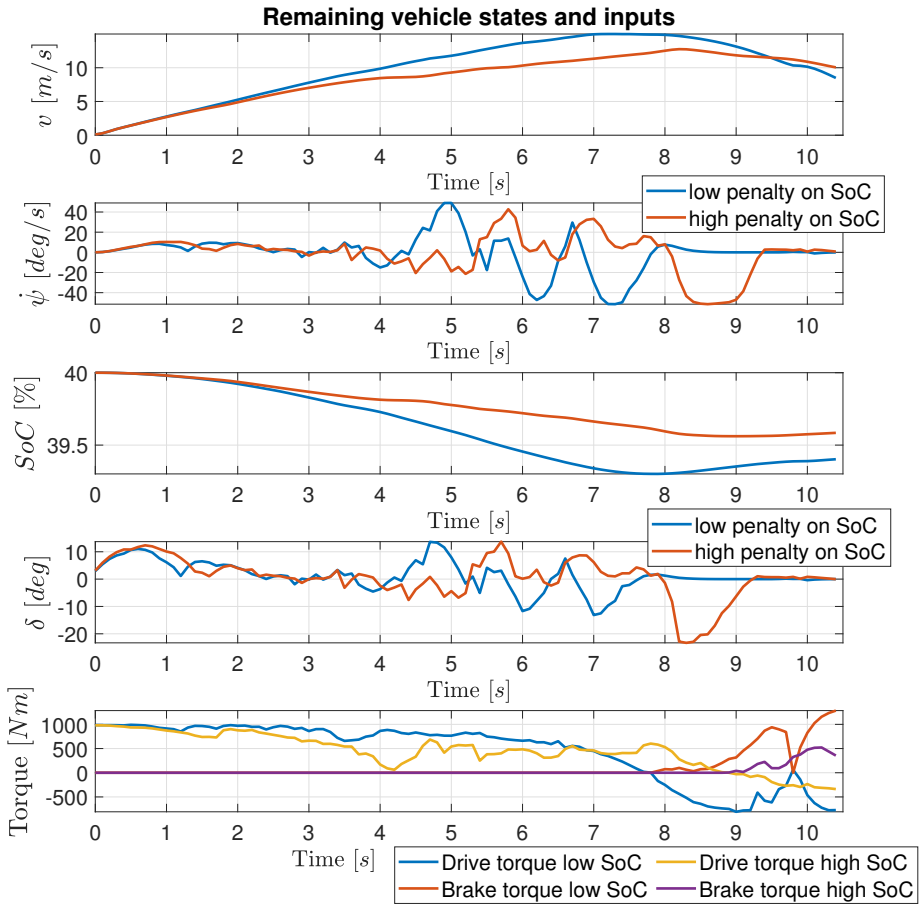
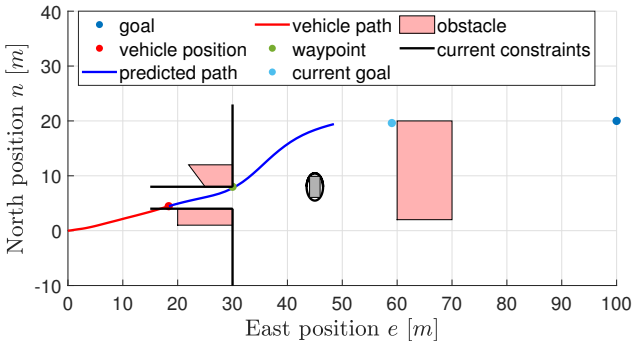
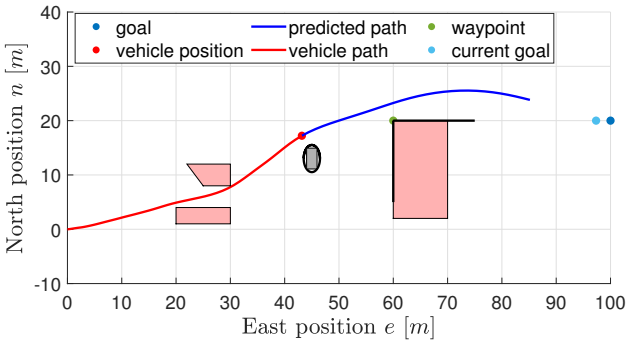


Figure 4.16: Other vehicle states and inputs in the second scenario. Two versions of cost function compared - with low and with high penalty in the cost function for the state of charge maximization.

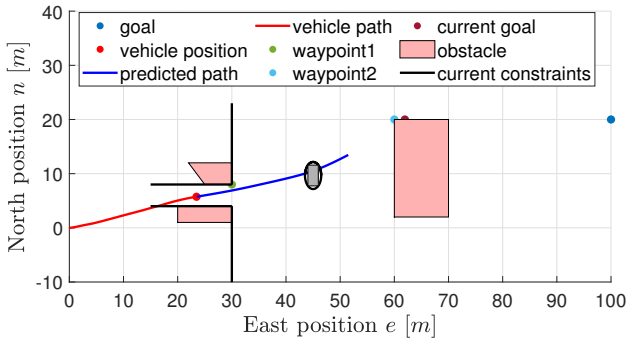
The vehicle is not required to completely stop at the goal position as this is impossible from the higher velocities within the 3 seconds long prediction horizon. From the subfigure which displays SoC, it can be seen that the trajectory with higher penalty on the state of charge maximization ends with higher SoC value (in addition, it is the value before the main usage of regenerative braking as the vehicle does not reach the goal yet (see Fig. 4.15)). The last subfigure shows that smaller drive torque is used during the ride for the trajectory with high SoC penalty. The steering angle δ is not so smooth as in the previous scenario. The matrix $R_{d_\delta} = diag(20)$ is used, as this slightly smaller cost for δ improves the searching in the space with constraints given by obstacles. However, the main reasons for the rough course of steering angle are close passing around obstacles and design model vs. high fidelity model mismatch.



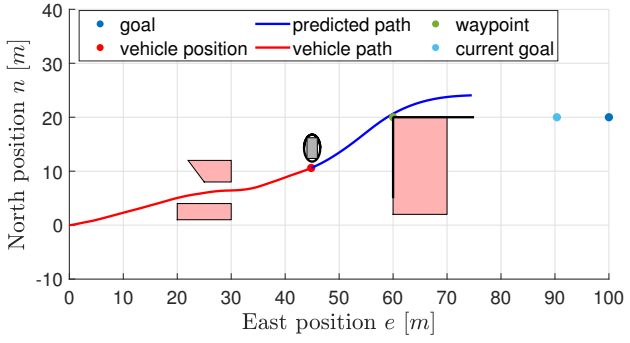
(a) : Predicted trajectory - low penalty on SoC maximization.



(b) : Predicted trajectory - low penalty on SoC maximization.



(c) : Predicted trajectory - high penalty on SoC maximization.



(d) : Predicted trajectory - high penalty on SoC maximization.

Figure 4.17: Predicted trajectories by NMPC for both situations - with low (4.17a, 4.17b) and high (4.17c,4.17d) penalty on SoC maximization.

4.4.3 Scenario 3

The last scenario is slightly different as it requires path (given by positions) tracking. The desired path is given by the center line of the right lane. Moving obstacle is in the right lane and goes with velocity $3\,\frac{\mathrm{m}}{\mathrm{s}}$. Again, the NMPC cost is formulated as 4.48. The position references are given to all the samples in the prediction horizon as a points which are possible to reach by vehicle in the time given by position of the sample in the prediction horizon. The matrices Q_{r_e} and Q_{r_n} which weights position reference tracking are set such that they weight the close samples with bigger values and far samples with smaller values: $Q_{r_e} = Q_{r_n} = diag(0.4, .., 0.1)$

Requirements formulated by linear temporal logic are

$$\square\neg(\text{collision with moving obstacle}), \tag{4.51}$$

$$\square(\text{on the road}). \tag{4.52}$$

The requirements are reformulated to MPC constraints by soft output constraints for obstacles restrictions and road restrictions.

In the figure 4.18 the vehicle path is shown. The vehicle does not track the center line of the right lane exactly, rather it maintains its position in the left half of the right lane, then it overtakes the moving obstacle and continues.

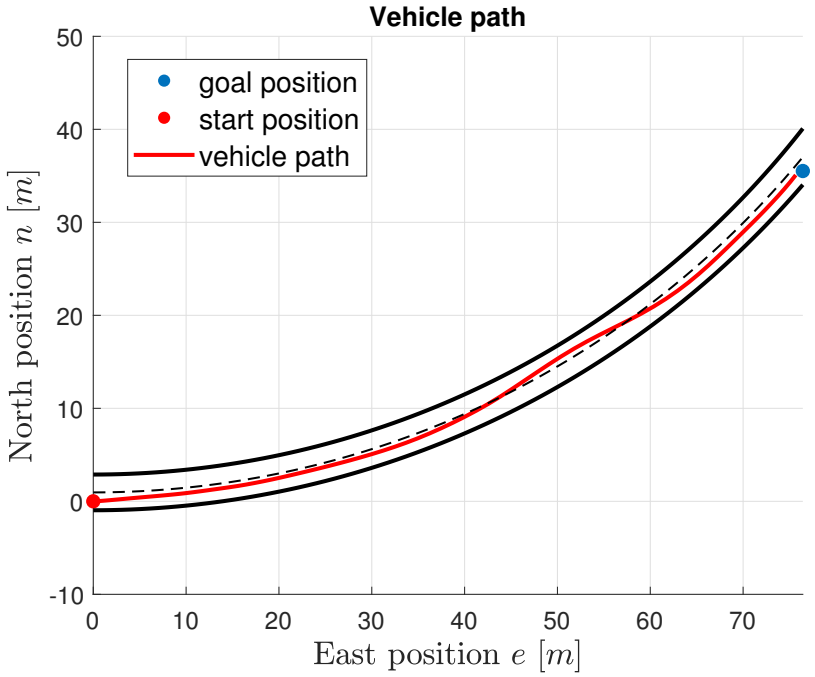


Figure 4.18: Vehicle path by NMPC in the last scenario - overtaking

Moving obstacle is shown in the predictions for certain simulation times in the figure 4.20 for a better understanding of the described situation.

The vehicle starts from initial states $v_{init} = 0.1\,\frac{\mathrm{m}}{\mathrm{s}}$, $\psi_{init} = 0\,\mathrm{deg}$, $SoC_{init} = 40\,\%$. From sub-figure which shows steering angle δ of the figure 4.19 can be seen that at the beginning the vehicle turns slightly (look at the $\dot{\psi}$, because of the low velocity) to the left, because of the weights on the position references.

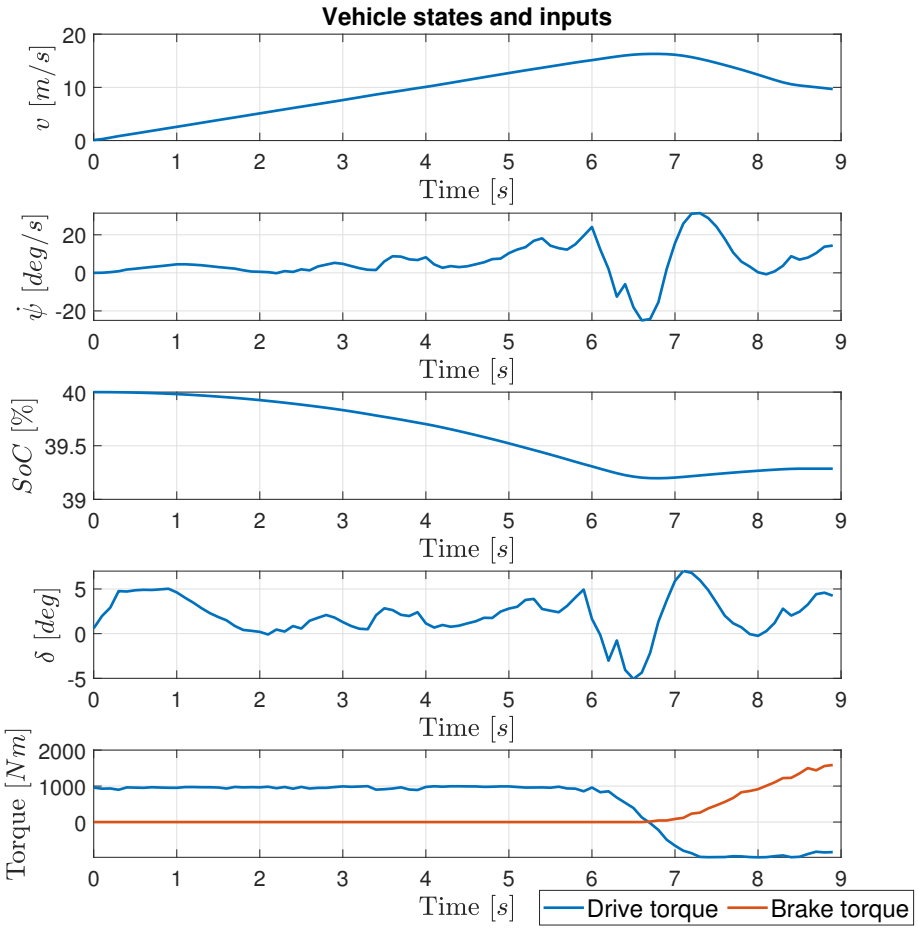
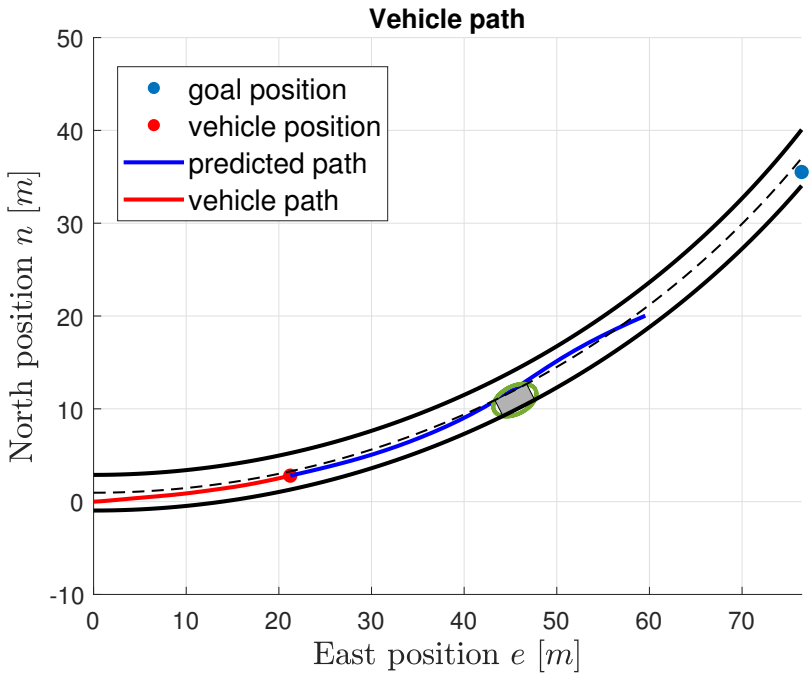
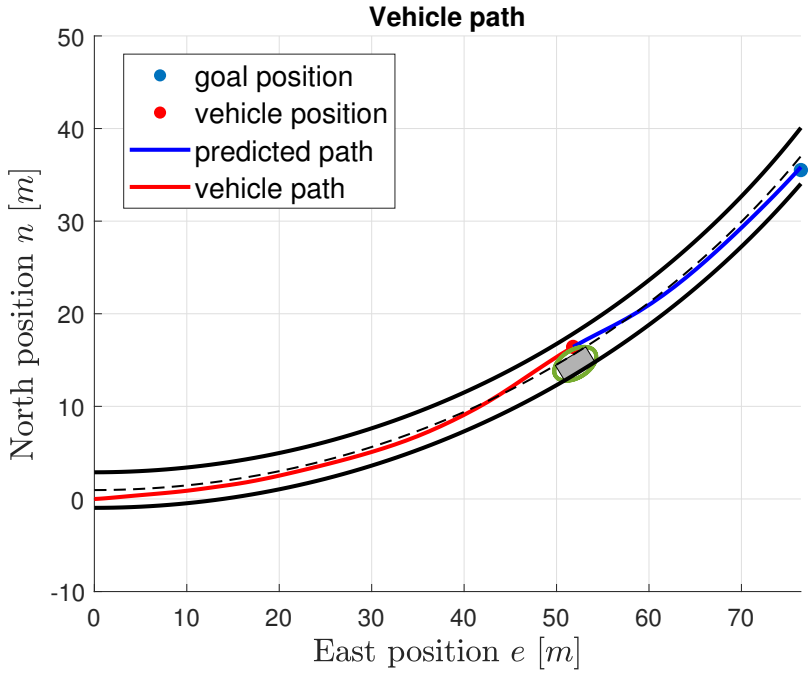


Figure 4.19: Other vehicle states and inputs in the last scenario.



(a) : Predicted trajectory in scenario 3.



(b) : Predicted trajectory in scenario 3.

Figure 4.20: Predicted trajectory by NMPC in the last scenario for different times.

Chapter 5

Minimum-violation Planning

5.1 MVP introduction

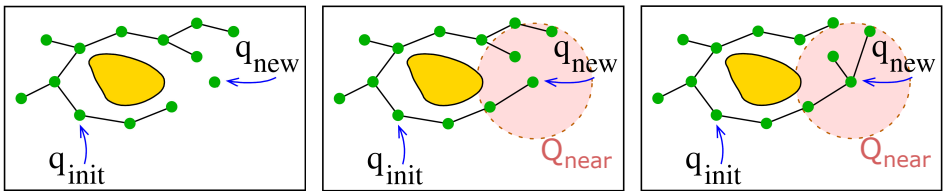
In this section, the algorithm named minimum-violation planning (MVP) will be concisely described. Unlike model predictive control, the MVP is a recent and novel approach. The algorithm state of the art is given in [25]. Therefore, the paper is the main source of information about MVP for the thesis. Further information, providing also the algorithm development, can be found in prior articles [23], [21], [26], [4], [17], in which the problem was solved via automata theory. However, this approach was leaved later in favor of sampling algorithms. As the name suggests, the algorithm is developed for path planning problems. Extension to all states generating trajectory is given in MVP. Phrase minimum-violation indicates the trajectory is in some sense optimal. The optimum is given by a cost function minimum, where for MVP the cost is a vector function, whose dimensions are equivalent to levels with different priorities.

MVP is sampling-based algorithm. States of the system are randomly sampled and connected. For this purpose, algorithms like Rapidly-exploring random tree (RRT), Rapidly-exploring random graph (RRG) or Probabilistic roadmap (PRM) are considered.

The following MVP algorithm overview is adopted from [25]. It is repeated here and described for clarity.

Algorithm 3 Minimum-violation planning. (Adopted from [25].)

1: $Q_K \leftarrow \emptyset$; $R_K \leftarrow \emptyset$; $Q_{goal,K} \leftarrow \emptyset$;
2: add(q_{init}) to Q_K;
3: **for all** $i \in \mathbb{N}_{\leq n}$ **do**
4: $q_{new} \leftarrow$ sample(i);
5: Q_{near} = near(q_{new});
6: add(q_{new}) to Q_K;
7: **for all** $q \in Q_{near}$ **do**
8: **if** steer(q, q_{new}) $\neq \emptyset$ **then**
9: connect(q, q_{new});
10: **for all** $q \in Q_{near}$ **do**
11: **if** steer(q_{new}, q) $\neq \emptyset$ **then**
12: connect(q_{new}, q);
13: **if** $q_{new} \in Q_{goal}$ **then**
14: $Q_{goal,K} = Q_{goal,K} \cup q_{new}$;
15: **return** $K_n = (Q_K, q_{init}, R_K, \Pi, L, W_K), Q_{goal,K}$



(a) : Line 4. **(b) :** Lines 7 - 9. **(c) :** Lines 10 - 12.

Figure 5.1: RRT* for Minimum-violation planning. Adopted from [24] and modified.

The algorithm 3 will be briefly described here, but for deeper understanding the reader is referred to [25] (page 5). At the start, a set of sampled states Q_K, set of existing transitions between states R_K and set of near to goal states $Q_{goal,K}$ are initialized to empty sets. Initial state is added to set Q_K in function *add* at line 2. Then a graph with n nodes is constructed. New node (an independent, identically distributed sample from state space uniform distribution) is added to Q_K in each iteration. Its nearest neighbors in Q_K (given by some state distance limit) are found (line 5) and for each neighbor trajectory from the neighbor to new node q_{new} is constructed (8). If it exists, the node is connected to the neighbor (line 9). *Connect* part of the algorithm differs with sampling-based algorithm used. RRG just adds the transition (q, q_{new}) to R_K and computes the cost $W_K(q, q_{new})$ from q to q_{new}. On the other hand, RRT computes the transition cost $W_K(q, q_{new})$ and cost $J_K(q)$ from root node q_{init} to q, then transition is added only if it improves the cost $J_K(q_{new}) := J_K(q) + W_K(q, q_{new})$ (previous transition to node q_{new} is deleted if there was any). After that, each neighbor trajectory is searched and the

same connection procedure is done in reverse order from q_{new} to q (lines 10 - 12). Finally, q_{new} is added to $Q_{goal,K}$ if it is in goal states Q_{goal}. The result is a graph (RRG) or tree graph (RRT) with nodes Q_K, transitions R_K, transition costs W_K, initial node q_{init} and set $Q_{goal,K}$ matching goal states. The tuple $K_n = (Q_K, q_{init}, R_K, \Pi_K, L_K, W_K)$ is called weighted Kripke structure, where Π_K is a set of atomic propositions and $L_K : Q_K \to 2^{\Pi_K}$ is labeling function assigning each state set of atomic propositions that are true at that state (2^{Π_K} is powerset of Π). These concepts are important for the computation of the cost function W_K and will be further clarified.

5.1.1 Linear temporal logic

Trajectory which is searched has to minimize the cost function $J_K(q_{goal})$ of the trajectory from q_{init} to $q_{goal} \in Q_{goal,K}$. The cost function is in MVP formulated based on rules given by Linear temporal logic (LTL), more precisely by Finite linear temporal logic (FLTL). FLTL formulas are constructed from

- set of atomic propositions Π,
- logic operator negation ($\neg$) and disjunction ($\vee$),
- temporal operator next ($\bigcirc$) and until (U).

Further logic operators, eg. conjunction ($\wedge$), implication ($\Longrightarrow$), equivalence ($\Longleftrightarrow$), $true$, $false$, can be constructed from fundamental operators and further temporal operators, eg. always ($\Box$), eventually ($\Diamond$), can be constructed from fundamental operators. FLTL formula is given inductively as

- p is FLTL formula if $p \in \Pi$,
- if p, p' are FLTL formulas then $\neg p$, $p \vee p'$, $\bigcirc p$, $p\,U p'$ are FLTL formulas.

FLTL will be restricted to a smaller class called $si - FLTL_{G_x}$. Restriction is done to better accommodate algorithm 3 cost function W_K computation. The computation of W_K (in *connect* part of the algorithm) cares only about a state and its nearest parent state and the relevant transition. With FLTL, a much bigger class of formulas can be built up (eg. by using $\bigcirc \bigcirc p$ state and the parent of its parent can be related), therefore the smaller class of formulas is needed. $si - FLTL_{G_x}$ is defined as follows (adopted from [25]):

Definition 5.1. An $si - FLTL_{G_x}$ formula over a set Π of atomic propositions is an FLTL formula that is stutter-invariant (explained below) and is of the form

$$\varphi = \Box P_x, \tag{5.1}$$

where P_x belongs to the smallest set defined inductively by the following rules:

- p is a formula for all $p \in \Pi \cup \{True, False\}$,
- $\bigcirc p$ is a formula for all $p \in \Pi \cup \{True, False\}$, and
- if P_x^1 and P_x^2 are formulas, then so are $\neg P_x^1$, $P_x^1 \vee P_x^2$, $P_x^1 \wedge P_x^2$ and $P_x^1 \implies P_x^2$.

In other words, P_x is a Boolean combination of propositions from Π and expressions of the form $\bigcirc p$, where $p \in \Pi$.

It is good to say that besides $si - FLTL_{G_x}$ formulas the mentioned article [25] defines also $si - FLTL_G$ formulas. Every formula from $si - FLTL_{G_x}$ can be written in $si - FLTL_G$. The $si - FLTL_G$ nomenclature is then used through the paper. This is not so important for the thesis because it is used mainly for the proofs in the article.

Expressions for which will be evaluated their truth (interpreted) over FLTL formulas are called words. A finite word assigns a set of consecutive states $x = q_1 q_2 ... q_k$ labels $l_1 l_2 ... l_k$:

$$\omega(x) = l_1 l_2 ... l_k. \tag{5.2}$$

If the word ω satisfies formula φ it is written as $\omega \models \varphi$. As the system trajectory is continuous, for the MVP purpose, these states are taken only at times T, when function L_K has discontinuities:

$$T = \left\{ t \mid \lim_{t' \to t^-} L_K(x(t)) = \lim_{t' \to t^+} L_K(x(t)) \right\} \tag{5.3}$$

A word $\omega = l_1 l_2 ... l_k$ is called stutter-invariant if for any word $\omega' = l_1 l_2 ... l_{i-1} l_i l_i l_{i+1} ... l_k$ created from ω by duplicating some letters it applies $\omega' \models \varphi$ if and only if $\omega \models \varphi$. Additionally, ω' is called stutter-invariant because by removing some letters (word ω) it still applies $\omega' \models \varphi$ if and only if $\omega \models \varphi$.

5.1.2 Cost function formulation

The cost function is in MVP [25] given as a vector function

$$J(x) = [\lambda_P(\omega, \Psi_1), \lambda_P(\omega, \Psi_2), ..., \lambda_P(\omega, \Psi_N), \sum_i \rho_{1i} F_{1i}(s_s s_f), ..., \sum_j \rho_{mj} F_{mj}(s_s s_f)] \in \mathbb{R}^{N+m}. \tag{5.4}$$

Each dimension up to N in the cost function defines a different so-called level of unsafety λ_P of the word $\omega(x)$. Formulas $\varphi \in \Phi$ are organized into sets Ψ_i which are arranged in an ordered set

$$\Psi = (\Psi_1, \Psi_2, ..., \Psi_N), \tag{5.5}$$

where formulas in Ψ_1 have the biggest priority and Ψ_N the lowest. Each formula $\varphi \in \Phi$ has its own weight $\rho(\varphi)$, therefore unsafety of word $\omega(x)$ on level i is given by

$$\lambda_P(\omega, \Psi_i) = \sum_{\varphi \in \Psi_i} \rho(\varphi) \lambda(\omega, \varphi), \tag{5.6}$$

where $\lambda(\omega, \varphi)$ is level of unsafety of word $\omega = l_1 l_2 ... l_k$ of the formula φ defined as

$$\lambda(\omega, \varphi) = \sum_{i \in \mathbb{N}_{\leq n} \mid l_i l_{i+1} \not\models P_x} \begin{cases} t_{i+1} - t_i, & \text{if } l_i l' \not\models P_x \; \forall l' \in 2^{\Pi}, \\ 1, & \text{otherwise.} \end{cases} \tag{5.7}$$

This can differentiate between taking an unsafe transition and visiting an unsafe state (for more information see [25]).

The second part (dimensions $N+1$ to $N+m$) of the cost function 5.4 represents other trajectory costs by functions $F_{uv}(s_s s_f)$ ordered according to their priorities, where $s_s s_f$ denotes to the trajectory including inputs and times. Function $F(s_s s_f)$ can represent for example minimization of trajectory time, consumption or inputs magnitude. Only trajectory time minimization was stated in the paper [25], however, the paper mentions the ease of the extension like this one. The optimization is performed based on the cost function 5.4 dimension by dimension such that cost on the lower dimension has bigger priority. That is eventually possible to move functions F_l to lower dimensions to have bigger priority for them than for formulas in Ψ_k.

5.2 MVP vehicle design models

In this section, design models (control oriented models) of the vehicle for MVP algorithm are introduced at two slightly different levels of complexity. For

MVP algorithm, neither linearity nor convexity is required. Used functions can be very nonlinear, $if - else$ and similar structures are allowed. However, simplifications are needed as well due to computational complexity. For MVP algorithm, state complexity is needed to be reduced. *Steer* part of the MVP algorithm, where a trajectory from one state to another state is found, causes the main problem. This is time demanding and/or after some introduced adjustment memory demanding. This means it is not possible to use the high-fidelity model directly because it has too many states and the two-state connection would be almost impossible in a reasonable time. (More in section 5.3.) Because the states have been already reduced for both NMPC design models, the work is almost done. It is advantageous as algorithms comparison on the similar models can be done.

5.2.1 High complexity vehicle design model

The same model as in subsection 4.2.1 is used. The equations are not stated here again. The reader is referred to equations 4.7 - 4.30. The only difference is in Eq. 4.12 or 4.15 - $atan2$ function can be used here without any approximation as MVP is a derivative free method.

5.2.2 Low complexity vehicle design model

The same model as in subsection 4.2.2 is used. Again, the reader is refereed to equations 4.31 - 4.40. The equations are not stated here again. And no additional modification is done. This is advantageous because both algorithms can be compared on the same model.

5.3 MVP problem solution

This section describes modifications of the Minimum-violation planning (MVP) algorithm introduced in this work. Modifications are needed as direct usage of the algorithm is not possible for systems with a high number of states or inputs. In [25], Dubins car is used as an example. Dubins car has only three states x, y position and heading ψ. It goes with constant velocity and the only input is the angular velocity $\dot{\psi}$. Trajectory finding between two states is very simple for Dubins car model. That is the reason for its frequent

usage in path planning problems. A real car does not travel with constant velocity and etc. Thus, then the *steer* function in algorithm 3 leads to a complex two-point boundary value problem. This is the main bottleneck of the MVP algorithm. Modifications made on the following lines deal mainly with this problem. In the thesis, MVP is based on Rapidly-exploring random tree (RRT) and mainly on its asymptotically optimal variant RRT*. For other algorithms (like RRG), only some of the following modifications bring improvements.

5.3.1 Trajectories precomputation for MVP

This section deals with two-point boundary value problem (TPBVP) solved in each iteration of algorithm 3 at lines 8 and 11 for each node in the neighborhood. The TPBVP solution by optimization tools was considered. Specifically, CasADi optimization tool was used for trajectory solution. By CasADi one such TPBVP for MVP design models described at section 5.2 last about 10ms on a computer with parameters stated in appendix table B.5.. Solver Gurobi gave similar times. Thousands of nodes in the tree are needed for minimal coverage of environments similar to those presented in test scenarios. And if a neighborhood of 10-30 nodes is considered, the time consumed only by the TPBVP is in the order of minutes.

Therefore, a different technique is proposed. System trajectories are precomputed based on MVP design model for certain inputs, states and time grids. Then, the steer function in MVP algorithm searches for a suitable trajectory that starts in a given start state and ends near the end state of the TPBVP in precomputed trajectories. The procedure of trajectories precomputation is described in algorithm 4. Grids for inputs, states and for

Algorithm 4 precomputeTraj($inputsGrid, statesGrid, timeGrid$)

```
1: Inputs: inputsGrid, statesGrid, timeGrid
2: trajectories = [size(statesGrid), size(inputsGrid)]
3: for all state ∈ statesGrid do
4:     x_0 = state
5:     for all u ∈ inputsGrid do
6:         traj = integrate(x_0, u, timeGrid)
7:         add(traj) to trajectories[state]
   return trajectories
```

time are needed in the algorithm. Inputs and states grids are created by an appropriate step length between minimal and maximal inputs and states. Time grid is defined by a step length and is limited by a selected time. For example, the time step can be 0.05 s and the maximal time 2 s. A trajectory

is computed by integration for every state and every input in the grid. The integration starts from a state in the grid and is performed by application of (constant) input in the grid for a maximal time. Result of integration is the trajectory. Purpose of the time step in the grid is following: In the steer function, trajectories lasting less than the maximal time can be also taken into account when searching for the suitable trajectory in MVP. Thus, the states pertaining to these times in the time grid can be also taken as endpoints of trajectory. The trajectories are saved in KDtrees. KDtrees were implemented because it improves the query time. Complexity of the KDtree query is log(n) [1].

Not all states have to be included in the *statesGrid*. Under certain assumptions - like considering the same slope of the road everywhere - trajectory can be transformed from the initial point $(e, n, \psi) = (0, 0, 0)$ to arbitrary point (e, n, ψ) in the environment by well-known geometric transformation relations. Or under other assumption - slope of the road is not excessively changing within the *timeGrid* - the slope can serve for a grid. There is a lot of other possibilities that can be convenient to use and it depends on the given situation. These transformations can significantly reduce the time and memory demands for trajectories precomputation and reduce query time in the MVP.

The *steer* function with precomputed trajectories is given by algorithm 5. First, the nearest *state* to the start state q_{start} in the *statesGrid* is found. Then KDtree searching for a trajectory starting at *state* and ending near q_{end} is performed (line 2). Trajectory is accepted if the distance between q_{end} and end of the best trajectory is less than a specified threshold. Every point given by the time grid as described above can be viewed as an end point of some trajectory. Trajectory is then taken only to that point. If needed, transformation of the trajectory is performed in the *steer* function.

Algorithm 5 steer(q_{start}, q_{end})

1: $state$ = findNearest($q_{start}, statesGrid$)
2: $traj, traj_{end}$ = findNearestTraj($q_{end}, trajectories[state]$)
3: **if** distance($q_{end}, traj_{end}$) $\leq threshold$ **then**
4: **return** $traj$
5: **else**
6: **return** $\emptyset$

MVP with precomputed trajectories is further modified in algorithm 6. New sample (line 15) is added to the tree only if it has a parent. It was not so in the original MVP and for larger systems (larger state space) it led to situations where lots of nodes were not connected to any other node.

Algorithm 6 Minimum-violation planning with precomputed trajectories.

Inputs: $trajectories, statesGrid, timeGrid$
$Q_K \leftarrow \emptyset$; $R_K \leftarrow \emptyset$; $Q_{goal,K} \leftarrow \emptyset$;
add(q_{init}) to Q_K;
$i \leftarrow 0$; $val \leftarrow$ **false**
while $i \leq n$ **do**
    $q_{new} \leftarrow$ sample();
    $Q_{near} =$ near(q_{new});
    $J_K(q_{new}) =$ inf
    for all $q \in Q_{near}$ **do**
        if steer(q, q_{new}) $\neq \emptyset$ **then** ▷ *steer* - Algorithm 5
            $val =$ **true**
            connect(q, q_{new}) ▷ Algorithm 7
    if not val **then**
        continue;
    add(q_{new}) to Q_K;
    for all $q \in Q_{near}$ **do**
        if steer(q_{new}, q) $\neq \emptyset$ **then** ▷ *steer* - Algorithm 5
            connect(q_{new}, q) ▷ Algorithm 7
    if $q_{new} \in Q_{goal}$ **then**
        $Q_{goal,K} = Q_{goal,K} \cup q_{new}$;
    $i \leftarrow i + 1$
return $K_n = (Q_K, q_{init}, R_K, \Pi, L, W_K), Q_{goal,K}$

Algorithm 7 connect(q_{start}, q_{end})

if $J_K(q_{start}) + W_K(q_{start}, q_{end}) \leq J_K(q_{end})$ **then**
    $J_K(q_{end}) = J_K(q_{start}) + W_K(q_{start}, q_{end})$
    $R_K \leftarrow (R_K \setminus \{(q_1, q_{end}) \in R_K\}) \cup \{(q_{start}, q_{end})\}$

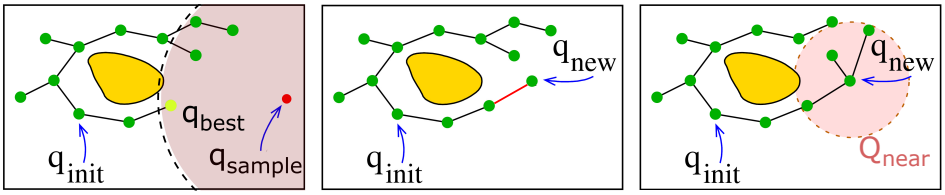
5.3.2 Modification to RRT* for MVP - version 1

Unfortunately, a trajectory between two states (produced by *steer* function) often does not exist, which leads to discarding the sampled state. There are multiple reasons. At first, generally for a system like vehicle, the closer are the states in eg. position, the closer must be their velocities, heading, etc., for a connection to exist. Furthermore, if nodes are too far from each other, a connection also does not exist due to the maximal time in the grid. Finally, the grid for trajectory precomputation cannot be too dense as it leads to time and memory issues. And it can cause a missing trajectory, although in the real world (or by some optimization tools) a trajectory would exist. The problem is also found when the trajectory exists only for a few nodes (one, two) in the neighborhood. Then, although the chosen node can have the best

cost from these, the cost can be still much worse compared to other nodes in the neighborhood for which a trajectory was not found. Then the algorithm produces possibly suboptimal connections between states rather than the best connections based on the cost optimization. Following proposed modification to RRT* introduced in algorithm 8 can deal with the issues described above.

Algorithm 8 Minimum-violation planning - version 1.

1: **Inputs:** $trajectories, statesGrid, timeGrid$
2: $Q_K \leftarrow \emptyset$; $R_K \leftarrow \emptyset$; $Q_{goal,K} \leftarrow \emptyset$; $i \leftarrow 0$;
3: add(q_{init}) to Q_K;
4: **while** $i \leq n$ **do**
5: $q_{sample} \leftarrow$ sample();
6: $Q_{near} =$ near(q_{sample});
7: $q_{best} =$ bestNode(Q_{near})
8: $q_{new} =$ chooseRandomTraj($q_{best}, trajectories$)
9: $J_K(q_{new}) =$ inf
10: connect(q_{best}, q_{new}) ▷ Algorithm 7
11: add(q_{new}) to Q_K;
12: $Q_{near} =$ near(q_{new});
13: **for all** $q \in Q_{near}$ **do**
14: **if** steer(q_{new}, q) $\neq \emptyset$ **then** ▷ *steer* - Algorithm 5
15: connect(q_{new}, q) ▷ Algorithm 7
16: **if** $q_{new} \in Q_{goal}$ **then**
17: $Q_{goal,K} = Q_{goal,K} \cup q_{new}$;
18: $i \leftarrow i + 1$
19: **return** $K_n = (Q_K, q_{init}, R_K, \Pi, L, W_K), Q_{goal,K}$



(a) : Lines 5 - 7. **(b) :** Lines 8 - 11. **(c) :** Lines 12 - 15.

Figure 5.2: Modified RRT* for Minimum-violation planning. (Figures inspired by [24] and modified.)

Here, the state space is sampled (q_{sample}) only for neighborhood selection. Then the node q_{best} with the best cost from the neighborhood is selected and from this node a random trajectory is chosen. New state q_{new}, the end of the trajectory, is added to the set of states. Finally, reconnecting as in the original MVP is done nevertheless with precomputed trajectories.

The aim of the adjusted part of the algorithm was to produce similar results as produced by the original RRT*. Random sampling ensures a random choice of the state space as in the original algorithm. Then the added state q_{new} is roughly in a similar part of the state space (can be slightly different). Although this modification brings a lot of pros, it has also cons. In the original MVP, the best node in the neighborhood can be chosen also based on the cost given by transition between the node and the currently sampled node. In the modified version 1, the transition cost is not known at the time when the best node q_{best} is chosen from the neighborhood and therefore it is assumed in advance that the transition cost is much smaller than the cost of the particular node (node cost is derived from the LTL rules). Solution to this small imperfection is proposed in the next subsection and can be also applied with small modifications to this version of the MVP algorithm.

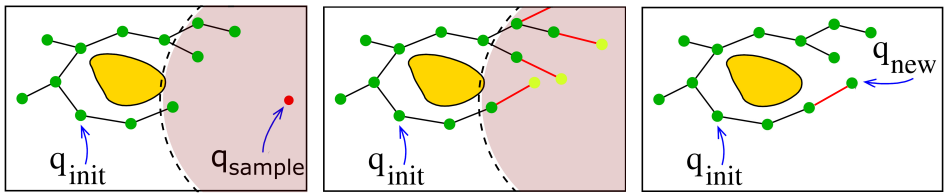
5.3.3 MVP based on modified RRT for large systems - version 2

Trajectories precomputation is not realistic for large systems because of the time and memory demands. MVP algorithm based on RRT is proposed for systems with a large number of states and/or inputs in this subsection (algorithm 9). Unlike RRT*, RRT is not asymptotically optimal. It is only probabilistic complete. RRT does not include the last reconnection part of the RRT*. If the reconnection part is dropped from the previous proposed version 1 (section 5.3.2), the trajectory precomputation is not needed anymore. Except that, the algorithm can remain unchanged, only random inputs are chosen at line 6 and integrated, instead of random trajectory choosing in the previous version 1.

Promised solution to the imperfection mentioned at the end of the previous subsection is following. Sample q_{sample} from the state space is taken. It serves only for neighborhood selection. Then a random input is chosen. The input is applied to all nodes (or to a certain number of nodes with the best costs from these) in the neighborhood and for each node the trajectory is obtained by integration. Finally, from the ends of the trajectories, node with the best cost q_{new} is added to the set of states. Here, the transition cost of two consecutive states can be included in the optimization, because the best node is chosen from the final (possibly added) states, not from the parents states as in the version 1. The approach can be used in the previous version 1, only random trajectories can be used instead of random inputs.

Algorithm 9 Minimum-violation planning - version 2.

1: $Q_K \leftarrow \emptyset$; $R_K \leftarrow \emptyset$; $Q_{goal,K} \leftarrow \emptyset$; $i \leftarrow 0$;
2: add(q_{init}) to Q_K;
3: **while** $i \leq n$ **do**
4: $q_{sample} \leftarrow$ sample();
5: $Q_{near} =$ near(q_{sample});
6: $u =$ chooseRandomInputs()
7: $J_K(q_{new}) =$ inf
8: **for all** $q \in Q_{near}$ **do**
9: $q_{aux} =$ integrate(q, u, t)
10: connect(q, q_{aux}, q_{new}) ▷ Algorithm 10
11: add(q_{new}) to Q_K;
12: **if** $q_{new} \in Q_{goal}$ **then**
13: $Q_{goal,K} = Q_{goal,K} \cup q_{new}$;
14: $i \leftarrow i + 1$
15: **return** $K_n = (Q_K, q_{init}, R_K, \Pi, L, W_K), Q_{goal,K}$



(a) : Lines 4 - 5. **(b) :** Lines 6 - 10. **(c) :** Line 11.

Figure 5.3: Modified RRT for Minimum-violation planning. (Figures inspired by [24] and modified.)

Algorithm 10 connect($q_{start}, q_{end}, q_{new}$)

1: **if** $J_K(q_{start}) + W_K(q_{start}, q_{end}) \leq J_K(q_{new})$ **then**
2: $J_K(q_{end}) = J_K(q_{start}) + W_K(q_{start}, q_{end})$
3: $R_K \leftarrow (R_K \setminus \{(q_1, q_{new}) \in R_K\}) \cup \{(q_{start}, q_{end})\}$
4: $q_{new} \leftarrow q_{end}$

5.4 MVP open loop results

Results of MVP method are presented in this section. Outcomes are given and described for scenarios (Fig. 2.1), (Fig. 2.2) and (Fig. 2.3) by MVP algorithm. Zero inclination angle $\alpha_{inc} = 0$ is considered for all scenarios. Code was implemented in Python. Presented results in this section are outcomes of the MVP algorithm based on the design models for one optimization

iteration by MVP. Thus, the optimization is run by MVP and the optimal inputs and states are returned, no simulation on the high-fidelity model is run. Low-complexity design model (section 5.2.2) and MVP version 1 (section 5.3.2) is used in the scenario 1 and 3, high-complexity design model (section 5.2.1) and MVP version 2 (section 5.3.3) is used in the scenario 2.

Input limits are the same as for NMPC algorithm 4.4. The high complexity design model cannot be used in the first proposed modified version 1 of MVP. Time demands for trajectories precomputation (in order of hours) and mainly memory demands that overflow RAM capacity (8 GB) are the reason. The demands for trajectories precomputation are achievable for the low complexity design model, if the brake input is not used. It is also possible with big steps in a brake grid, however it creates files with trajectories in the order of GB. Therefore, in the presented scenarios, the brake is not used together with version 1 of the MVP (braking is performed by motor torque). The version 2 is capable to run on both complexity levels of the design models with all inputs.

A lot of parameters have to be tuned for MVP, for example (there are many others that are not mentioned; typical values in the scenarios are stated, but note that these can vary for the presented scenarios or MVP versions)

- distance threshold in steer function for searching near nodes reconnections in precomputed trajectories: 1,
- steps in states grid for trajectory precomputation: eg. $v = 0.1\,\frac{\mathrm{m}}{\mathrm{s}}$,
- steps in inputs for trajectory precomputation: torques: $40\,\mathrm{N\,m}$, $\delta = [-25, ..., 0, 0.25, 1, 2.25, 4, 6.25, 9, 12.25, 16, 20.25, 25]$ deg,
- maximal time of the trajectories (transitions): $2\,\mathrm{s}$,
- time step for possible nodes on the trajectories (transitions): $0.1\,\mathrm{s}$,
- weights on the individual states (variables) for state distances computations: $\begin{bmatrix} e & n & \psi & v & SoC \end{bmatrix} = \begin{bmatrix} 1 & 1 & 4 & 1 & 10 \end{bmatrix}$,
- distance limit for near nodes: function of current number of nodes in the tree or,
  number of wanted nearest nodes: 20-30,
- number of best nodes from nearest nodes for version 2: 5-10,
- number of nodes for limitation of sampling of area sampled with many nodes already there,
- number of nodes in the tree.

5.4.1 Scenario 1

Obstacles are not in this scenario, therefore no formulas defined by LTL logic are needed. The results for two different cost functions will be stated, version 1 of MVP is used. In the following lines, the expression q_1q_2 is denoted to the trajectory from one node q_1 to another node q_2 including also the inputs at these nodes. For the first case, the cost function is formulated as

$$J(q_1q_2) = (\rho_T F_T(q_1q_2) + \rho_{\Delta u} F_{\Delta u}(q_1q_2)) = (F_T(q_1q_2) + 0.05F_{\Delta u}(q_1q_2)), \tag{5.8}$$

where the function $F_T(q_1q_2)$ gives time of the trajectory q_1q_2 and $F_{\Delta u}(q_1q_2)$ defined as

$$F_{\Delta u}(q_1q_2) = 100(\delta_{q_1} - \delta_{q_2})^2 + 0.1(T_{req,q_1} - T_{req,q_2})^2, \tag{5.9}$$

gives the cost for consecutive inputs. The defined cost function $J(q_1q_2)$ has only one level of unsafety (priority level). The parameters $\rho_T = 1$ and $\rho_{\Delta u} = 0.05$ weights between these costs. Tree was created by MVP based on this formulation. The tree nodes and transitions between nodes are shown in the figure 5.4. The best path and nodes based on the cost function are displayed. The vehicle starts from initial states $v_{init} = 2\,\frac{\mathrm{m}}{\mathrm{s}}$, $\psi_{init} = 0\,\mathrm{deg}$, $SoC_{init} = 40\,\%$. The acceptable goal states are given by minimal distance 4 m from the goal.

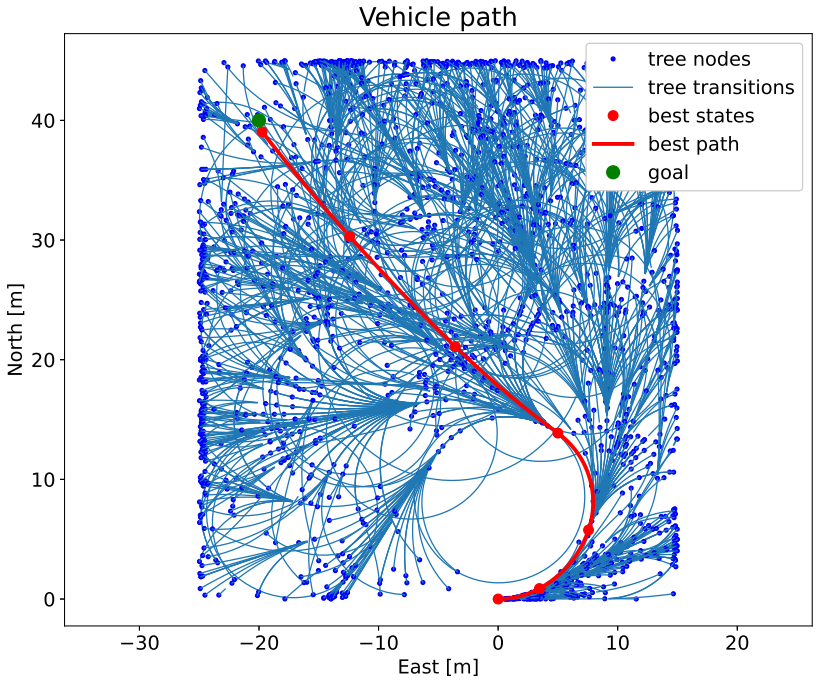


Figure 5.4: Tree by MVP and the best path based on the cost function 5.8.

Maximal turning radius can be seen in the figure around the start position. The maximal velocity was limited by 15 $\frac{\text{m}}{\text{s}}$. Other vehicle best trajectory states are shown in the figure 5.5. It is evident that the consecutive inputs for steering angle δ are very close as well as for torque inputs (maximal velocity is restrictive).

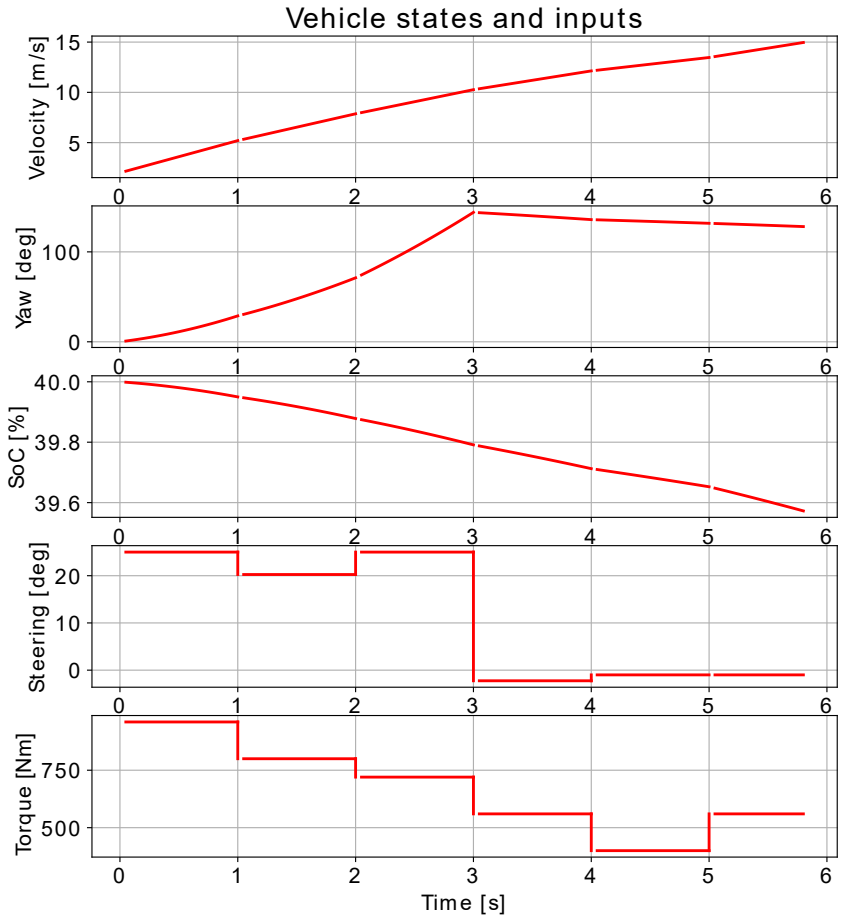


Figure 5.5: Vehicle states and inputs based on the cost function 5.8.

For the other case, the cost function is formulated as

$$\begin{aligned} J(q_1 q_2) &= (\rho_T F_T(q_1 q_2) + \rho_{soc} F_{soc}(q_1 q_2)) \\ &= (F_T(q_1 q_2) + 10 F_{soc}(q_1 q_2)), \end{aligned} \tag{5.10}$$

where the function $F_{soc}(q_1 q_2)$ defined as

$$F_{soc}(q_1 q_2) = SoC_{q_1} - SoC_{q_2}, \tag{5.11}$$

gives the cost for the state of charge maximization. The parameters are $\rho_T = 1$ and $\rho_{soc} = 10$. Tree was created by MVP based on this formulation. The tree nodes, transitions between nodes and the best path are shown in the figure 5.6.

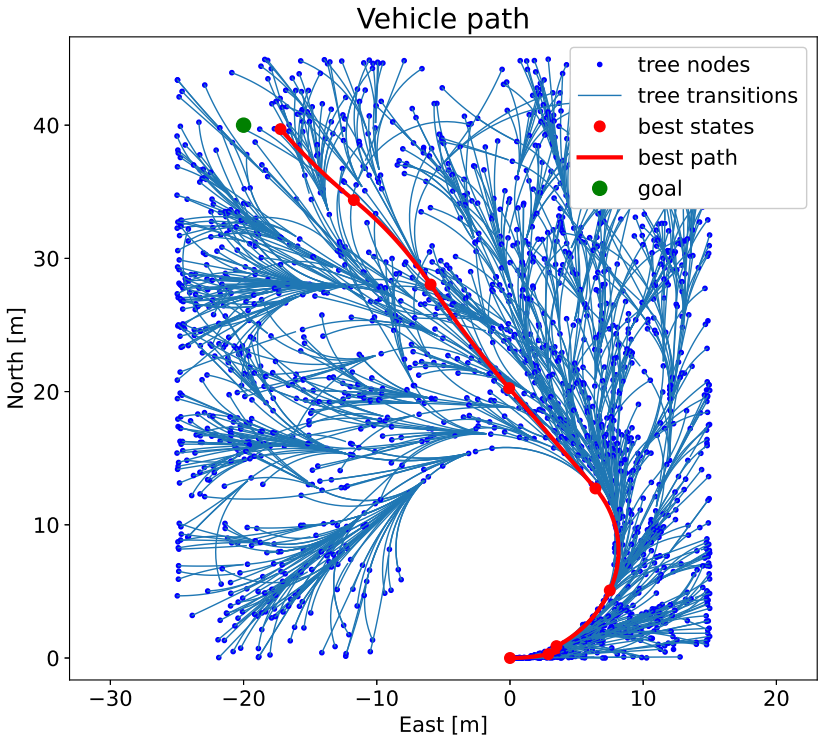


Figure 5.6: Created tree and best path based on the cost function 5.10.

Other vehicle states and inputs are shown in the figure 5.7. On first sight, it is clear that the consecutive inputs for both steering angle δ and drive torque are not so close as in the previous formulation. The formulation minimizes the consumption, which is shown in the subfigure denoted to the state of charge. The SoC ends with value above 39.85 % compared to the value under 39.6 % for the previous formulation. However, the end time is higher in this case.

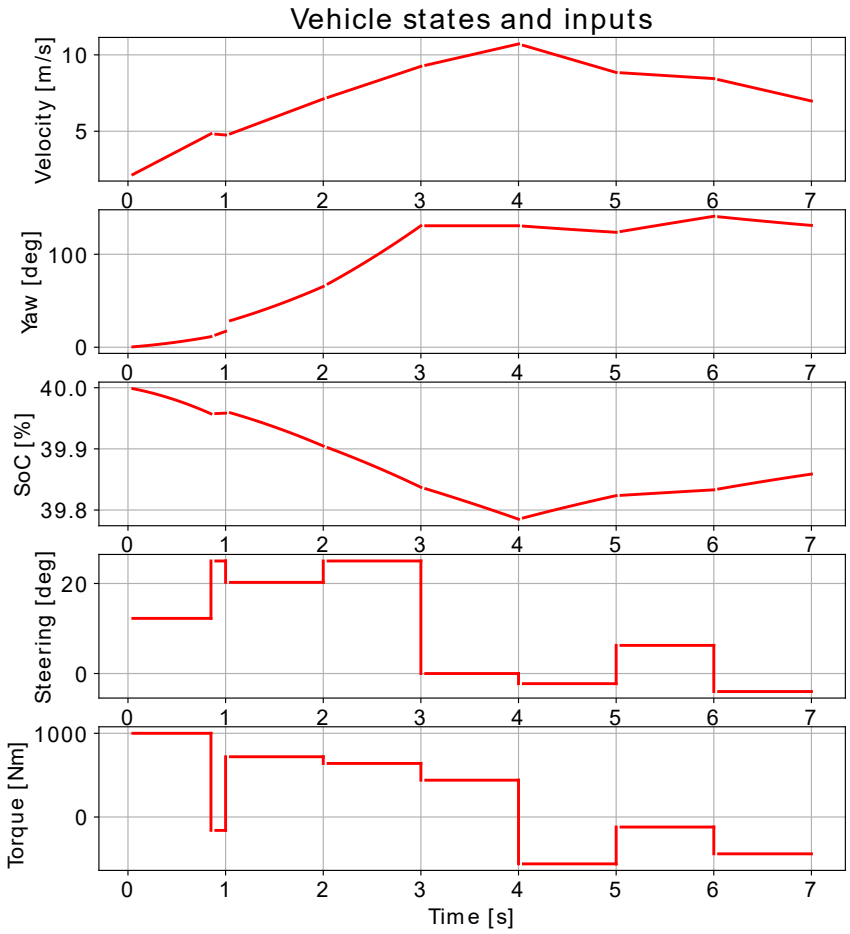


Figure 5.7: Vehicle states and inputs based on the cost function 5.10.

5.4.2 Scenario 2

For the second scenario, the requirements are formulated by linear temporal logic as

$$\varphi = \square\neg(\text{collision with obstacle}). \tag{5.12}$$

Then, the cost function formulation is as follows

$$\begin{aligned} J(q_1q_2) &= (\rho_\varphi(\varphi)\lambda(\omega(q_1q_2),\varphi), \rho_T F_T(q_1q_2)) \\ &= (\lambda(\omega(q_1q_2),\varphi), F_T(q_1q_2)). \end{aligned} \tag{5.13}$$

The cost function is two-dimensional, higher priority is given for collision avoidance with obstacles than for time minimization. Weights are set to $\rho_\varphi = 1$ and $\rho_T = 1$. Here, the scenario is without moving obstacle. The scenario with moving obstacle will be presented later when the feedback will be ensured and simulation will run.

The created tree with 2000 nodes is shown in the figure 5.8.The environment is not uniformly covered by samples. The situation can be different and the space around position $e = 35\,\mathrm{m}$, $n = 20\,\mathrm{m}$ can be sampled more densely in another run. Sampled nodes that are in the obstacles are chosen only with probability 30 %. It is possible as the version 2 without reconnecting part is presented. This can be disadvantageous for version 1 based on RRT* or in situations where the obstacles are, for example, only areas with grass or something that is passable.

The best trajectory is similar to the one got by NMPC. However, it is still a random-based algorithm and in other run trajectories with an upper path or with a path below the obstacle in the left bottom are got. A tree with more nodes will be needed to obtain a path really close to the optimal one.

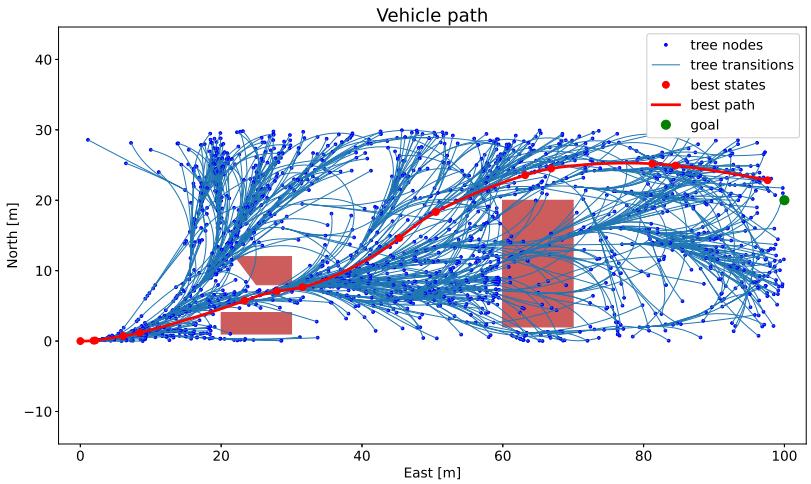


Figure 5.8: Created tree by MVP and the best path based on the cost function 5.13.

Other vehicle states and inputs for the scenario are in the figure 5.9.

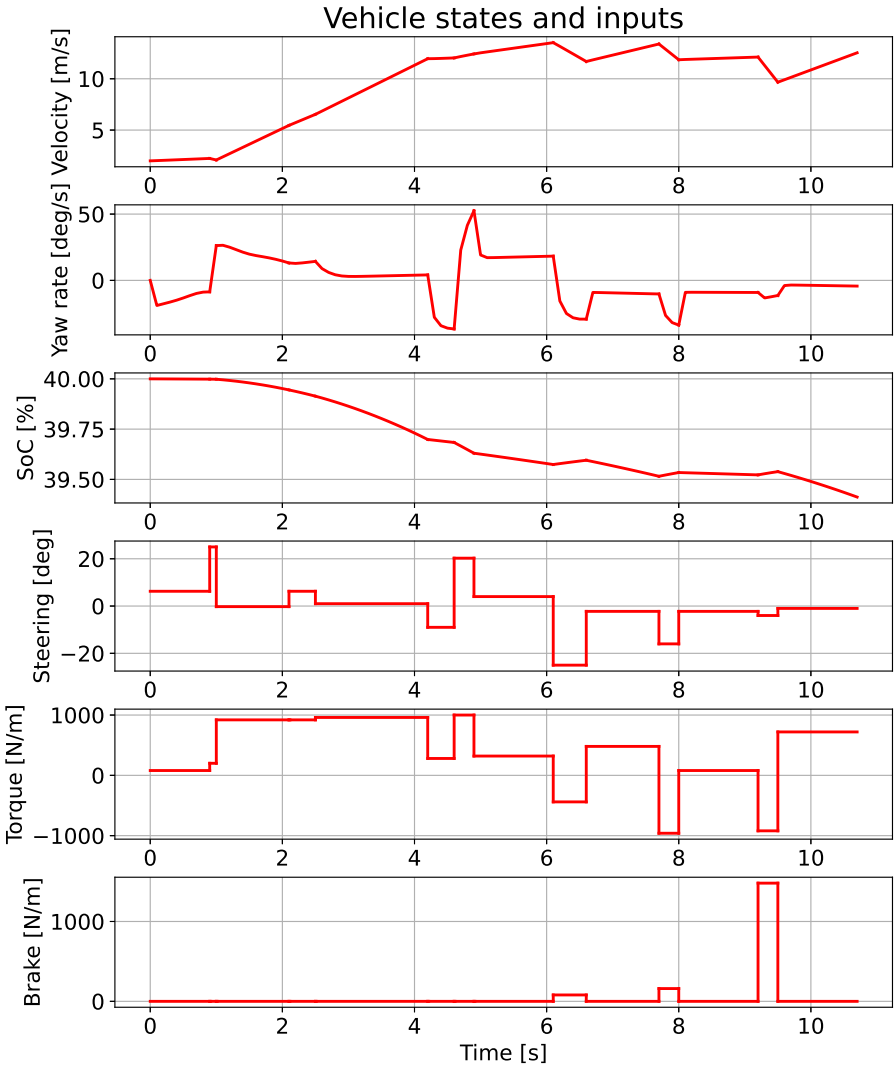


Figure 5.9: Vehicle states and inputs based on the cost function 5.13.

5.4.3 Scenario 3

This scenario is rather a tracking scenario. Requirements are formulated by

$$\varphi = \square\neg(\text{collision with moving obstacle}). \tag{5.14}$$

Cost function formulation is

$$\begin{aligned} J(q_1 q_2) &= (\rho(\varphi)\lambda(\omega(q_1 q_2), \varphi), \rho_T F_T(q_1 q_2) + \rho_{CL} F_{CL}(q_1 q_2)) \\ &= (\lambda(\omega(q_1 q_2), \varphi), F_T(q_1 q_2) + 0.1 F_{CL}(q_1 q_2)). \end{aligned} \tag{5.15}$$

The cost function is two-dimensional. Weights are set to $\rho = 1$, $\rho_T = 1$ and $\rho_{CL} = 0.1$. Function $F_{CL}(q_1 q_2)$ defines tracking of the right lane center line

as a sum of squared distances of the individual points on the trajectory q_1q_2 from the center line (CL).

$$F_{CL}(q_1q_2) = sum(distance(q_1q_2, CL)^2). \tag{5.16}$$

A tree with a thousand of nodes is needed to achieve sufficient path tracking. In the following figure 5.10 a tree with 10000 nodes is created. Figure 5.11 shows other vehicle states and inputs.

The computation time was about 50 s. The best path fairly track the reference. About position $e = 50$ m overtaking maneuver is performed.

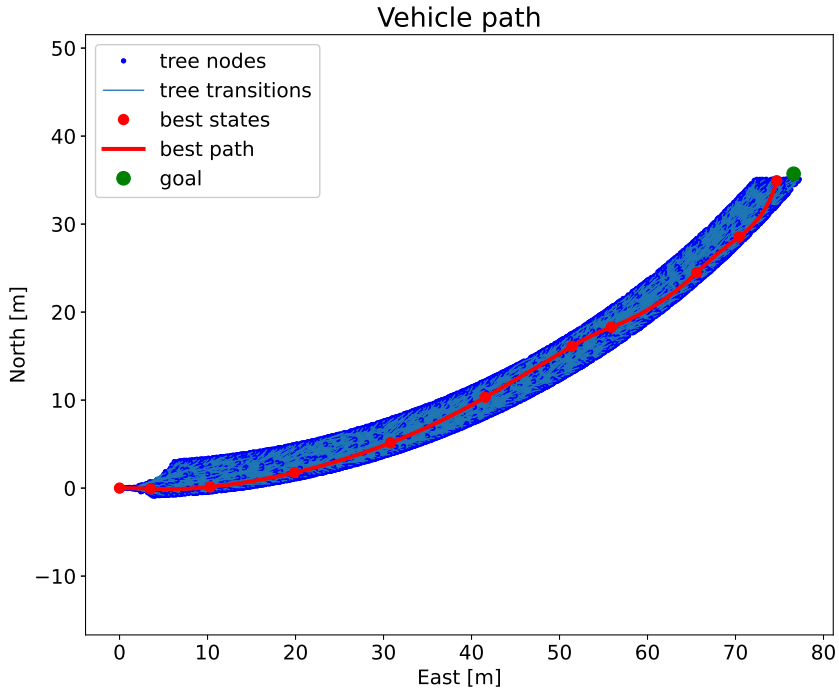


Figure 5.10: Created tree by MVP and the best path based on the cost function 5.15.

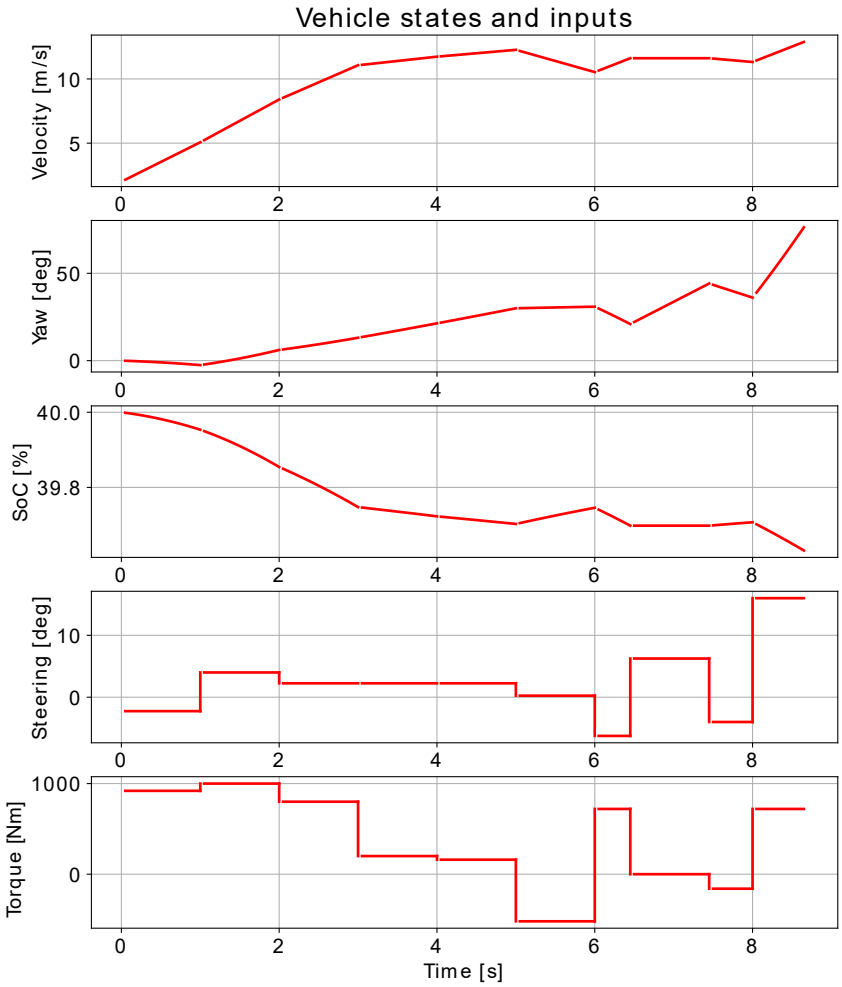


Figure 5.11: Vehicle states and inputs based on the cost function 5.15.

5.5 MVP in feedback

So far, MVP with the proposed various modifications is still an open-loop algorithm. Because the procedure consists only of trajectory planning at the start. Planned trajectory is based on design models and therefore it is needed to ensure feedback control due to model - reality mismatch.

There are two main approaches that can be used for feedback control

- trajectory tracking,

- trajectory generation (re-planning) in certain time steps (MPC - like approach).

Both approaches require knowledge about the current state of the system (at least those included in the design model). The knowledge about the state can be based on measurements or/and on estimation of the state.

5.5.1 Trajectory tracking

The trajectory tracking approach requires only one procedure of planning based on MVP at the beginning. When the best trajectory and the relevant inputs are acquired, a control that ensures tracking is run. There is a lot of options that can be used for control. For all, it can be mentioned that well-known and often used control by

- proportional–integral–derivative controller (PID),
- linear-quadratic regulator (LQR),
- model predictive control (MPC).

5.5.2 Trajectory replanning

The other option is replanning of the trajectory by MVP within particular time instants. Thus, this is the similar approach to MPC as described at the beginning of the chapter 4 with optimization by MVP. There can be various of techniques. One of the classification of the techniques can be based on the time step used:

(a) fixed time step less than or equal to the smallest time distance between consecutive states (nodes) in the tree

(b) variable time step given by the time distances between relevant consecutive states (nodes: root and its successor) in the best path.

The other classification can be based on the tree creation procedure **in every iteration**. We proposed:

(i) tree creation from scratch with root given by current system states,

(ii) part of the tree around the best path (all nodes that have the root first successor in their path to root) transformation to the new root given by the current system states, extension of the tree by adding new samples,

(iii) tree root transformation to the new root given by the current system states, leaving the tree as it is and rewiring the tree around the new root to the new root, possibly extension of the tree by adding new samples.

Note that in (iii) the root is transformed, whereas in (ii) the part of the tree is transformed. The variant (i) is easy for implementation, but it has higher time demands, because a new tree is created in each iteration. The variant (ii) is better from computational point of view as it keeps (slightly shift as required) the best part of the tree and therefore in situations when the time allocated for tree creation is not sufficient the good path is still provided. The tree is created along the previous best path and therefore it is probable that a slightly better path than the previous one in the previous best path neighborhood will be obtained. The variant (iii) keeps the whole tree and transforms only the root. The procedure of root rewiring in RRT* algorithm is proposed and explained in article [15]. The tree is extended uniformly in this variant, therefore if there is a better path than the one from the previous iteration in different part of the environment, the variant will likely give better results.

The variant (b)(ii) was chosen and implemented as a feedback control law. The procedure is following. At each iteration, when the tree is created, the best path, inputs and inputs duration is obtained and the first input is applied to the high-fidelity model for the respective time. If there was not a model - reality mismatch, the trajectory of the system would end in the first successor of the root in the best path. However, the mismatch is present, therefore the trajectory ends only near to that point. Thus, the best previous path, starting from the root's first successor is taken and all pertaining inputs are applied for relevant times to the design model starting from a new root. The new root is given by vehicle states. The obtained sequence of states similar to the previous path gives a foundation for tree creation in this iteration. It is a kind of warm started optimization problem.

Results of MVP for the previously described trajectory replanning in each iteration are given in the comparison chapter 6 for scenario 2.1 and 2.2. For the last scenario, this is skipped as it was shown that sufficient tracking requires thousands of nodes in the tree and it takes unusably much time. Moreover, the described procedure of trajectory replanning as MPC-like algorithm simulates the high-fidelity model for time given by the time distance between nodes in the path. It can be up to 2 s. There is a certain design model - reality

mismatch and the 2 s long open loop interval can cause trouble for reference tracking especially in the small environment like lane of the road.

The high-fidelity model is implemented only in MATLAB. Optimization by MVP is coded (as before) in Python. These two environments were connected. MATLAB code was launched from Python. Thus, inputs based on optimization by MVP are sent to MATLAB, then high-fidelity model is simulated for the needed time and the current vehicle state is fed back to Python.

Chapter 6

Algorithms Comparison

The methods NMPC and MVP are compared in this chapter. At first, the algorithms performances on the presented scenarios are discussed. Then a general comparison is given.

It could be illustrative to make a performance comparison based on values of the same cost functions. Unfortunately, the cost function formulations are too different. The cost for NMPC is one-dimensional, whereas the cost for MVP is multidimensional. However, mainly, the time minimization requirement is formulated as a distance minimization for NMPC and therefore cost of the resulting trajectory cannot be properly evaluated for NMPC and compared with the MVP trajectory cost afterwards.

6.1 Comparison in test scenarios

The comparison in scenarios 2.1 and 2.2 is given. The last scenario 2.3 is not discussed. The reason is given in subsection 5.5.2 (not usable for MVP).

6.1.1 Scenario 1

As an example, the results of algorithms for "similar" costs are discussed. Formulation of the NMPC cost is as it was in the NMPC results section,

$$\begin{aligned} J_{out,in} = \frac{1}{2}[&(e_{pred} - e_{ref})^T Q_{r_e}(e_{pred} - e_{ref}) + \\ &+ (n_{pred} - n_{ref})^T Q_{r_n}(n_{pred} - n_{ref}) + \\ &+ (SoC_{pred} - SoC_{ref})^T Q_{r_{SoC}}(SoC_{pred} - SoC_{ref})] + \\ &+ J_{in}. \end{aligned} \tag{6.1}$$

According to the proposed NMPC algorithm (section 4.3), only the current goal is given as a reference for the last sample in the prediction horizon (environment without obstacles) and the weight for it was chosen as $w = 0.1$ (last element on diagonal of both matrices Q_{r_e}, Q_{r_n}). State of charge reference is $SoC_{ref} = SoC + 2\%$ at the end of the prediction horizon. SoC is given by square matrix $Q_{r_{SoC}} = diag(0, .., 0, 10)$.

MVP cost function is formulated as

$$J(q_1 q_2) = (\rho_T F_T(q_1 q_2) + \rho_{soc} F_{soc}(q_1 q_2)) = (F_T(q_1 q_2) + 2F_{soc}(q_1 q_2)), \tag{6.2}$$

where

$$F_{soc}(q_1 q_2) = SoC_{q_1} - SoC_{q_2}, \tag{6.3}$$

gives the cost for the state of charge maximization. The parameters are $\rho_T = 1$ and $\rho_{soc} = 2$. The used MVP cost function is the same (except for ρ_{soc}) as in the open loop section for the case with time and state of charge minimization (formulation 5.10). The MVP final velocity state was required to be close to the earlier obtained final velocity by NMPC to have similar conditions. MVP tree was created in each iteration with 2000 nodes. The result of the closed-loop simulation is shown in figures 6.1, 6.2. The MVP path is a little wavy, it can be attributed to the longer intervals for the closed-loop structure (0.1 s for NMPC vs. up to 2 s for MVP) and to a randomness in the MVP algorithm.

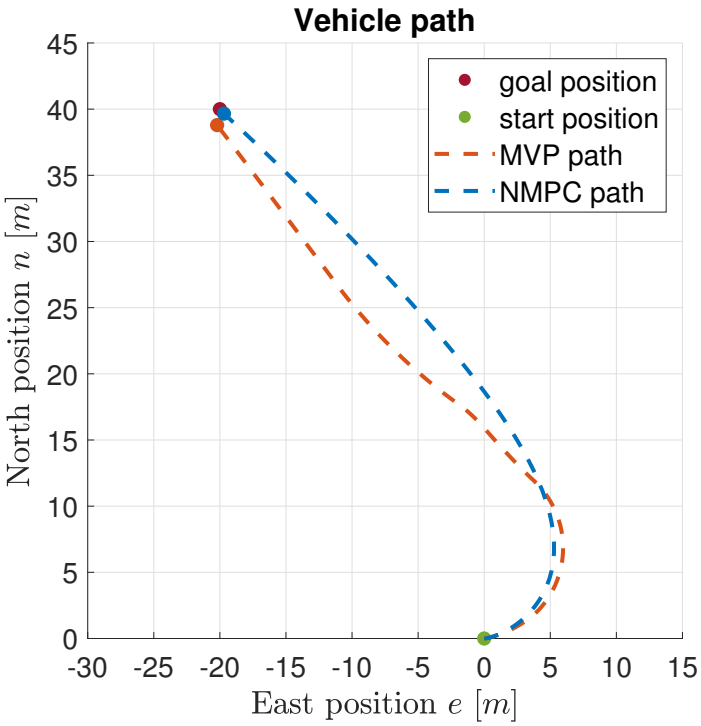


Figure 6.1: NMPC and MVP path comparison in the first scenario in feedback simulation.

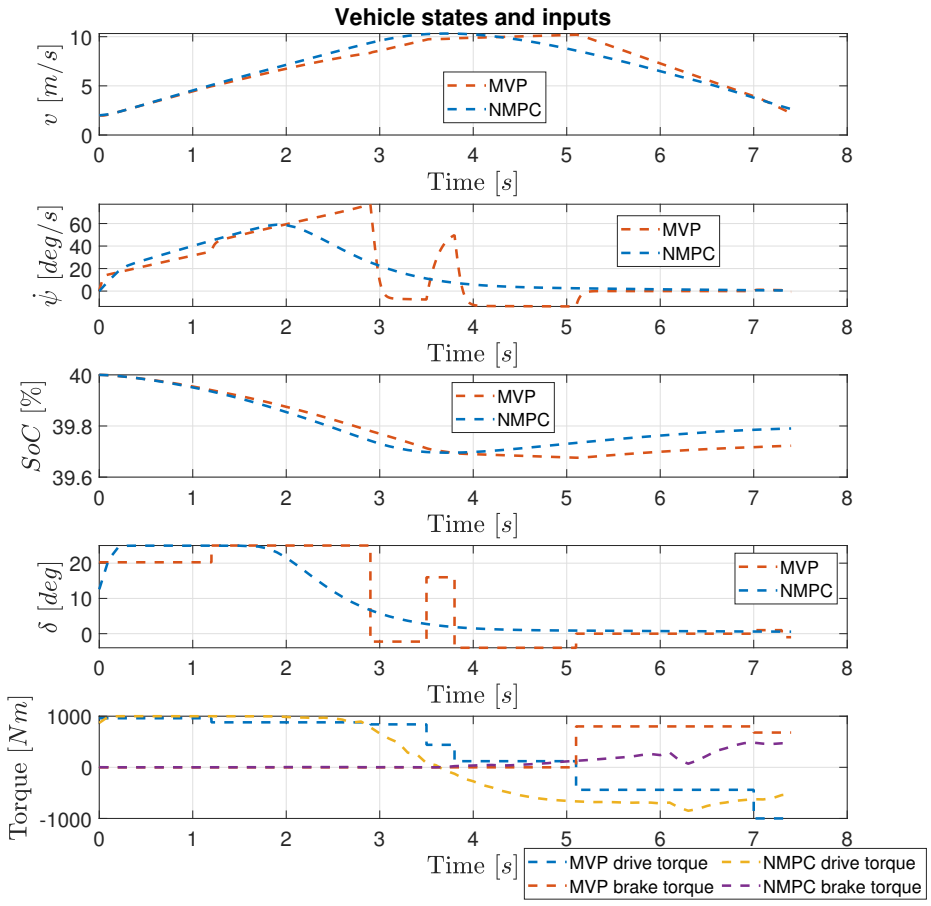


Figure 6.2: NMPC and MVP other states and inputs comparison in the first scenario in feedback simulation.

It can be seen that steering angle δ by NMPC is smooth. For MVP big steps in the steering angle δ are present, it is caused by a long interval between re-plannings. However, some steps (eg. about time 3.5 s) are caused by insufficient sampling or by insufficient (missing in this case - version 2 is used) reconnecting.

Both MVP and NMPC paths end in a similar position at the same time. Final SoC of NMPC is bigger than final SoC of MVP. Therefore, it is clear that NMPC algorithm has better performance in this case. Because the MVP is random-sampling based algorithm, its results vary. It is good to say that the cost of the presented MVP trajectory belongs rather to the better one obtained. Mostly higher end time or lower final SoC are got. Thus, it can be concluded that for this particular scenario NMPC gives better results.

6.1.2 Scenario 2

The obstacles (both static and moving) avoidance requirements are again formulated as

$$\varphi = \square\neg(\text{collision with obstacle}). \tag{6.4}$$

The NMPC cost function considered for the comparison is the same as used before 6.1, where matrices Q_{r_e} and Q_{r_n} which weights position reference tracking are set during the ride according to the proposed algorithm. Weight for the main reference sample is chosen as $w = 0.2$. Matrix $Q_{r_{SoC}}$ is set to $Q_{r_{SoC}} = diag(0, .., 0, 30)$. The same cost was used in NMPC section for this scenario. Obstacle avoidance requirement is reformulated to MPC soft output constraints.

MVP cost function formulation is as follows

$$\begin{aligned} J(q_1 q_2) &= (\rho_\varphi(\varphi)\lambda(\omega(q_1 q_2), \varphi), \rho_T F_T(q_1 q_2) + \rho_{soc} F_{soc}(q_1 q_2)) \\ &= (\lambda(\omega(q_1 q_2), \varphi), F_T(q_1 q_2) + 2F_{soc}(q_1 q_2)). \end{aligned} \tag{6.5}$$

MVP tree was created in each iteration with 2000 nodes. The result of the closed-loop simulation is shown in figures 6.3, 6.4. In this case, the vehicle controlled by MVP goes with the upper path (may differ simulation from simulation).

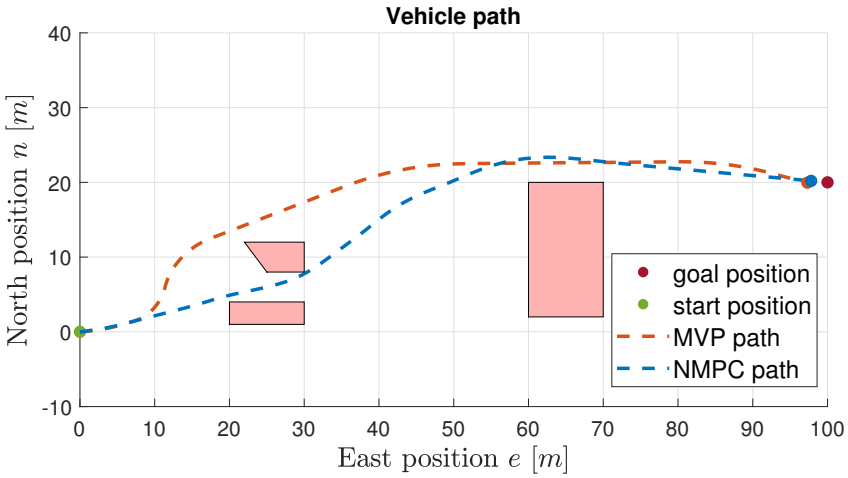


Figure 6.3: NMPC and MVP path comparison in the second scenario in feedback simulation.

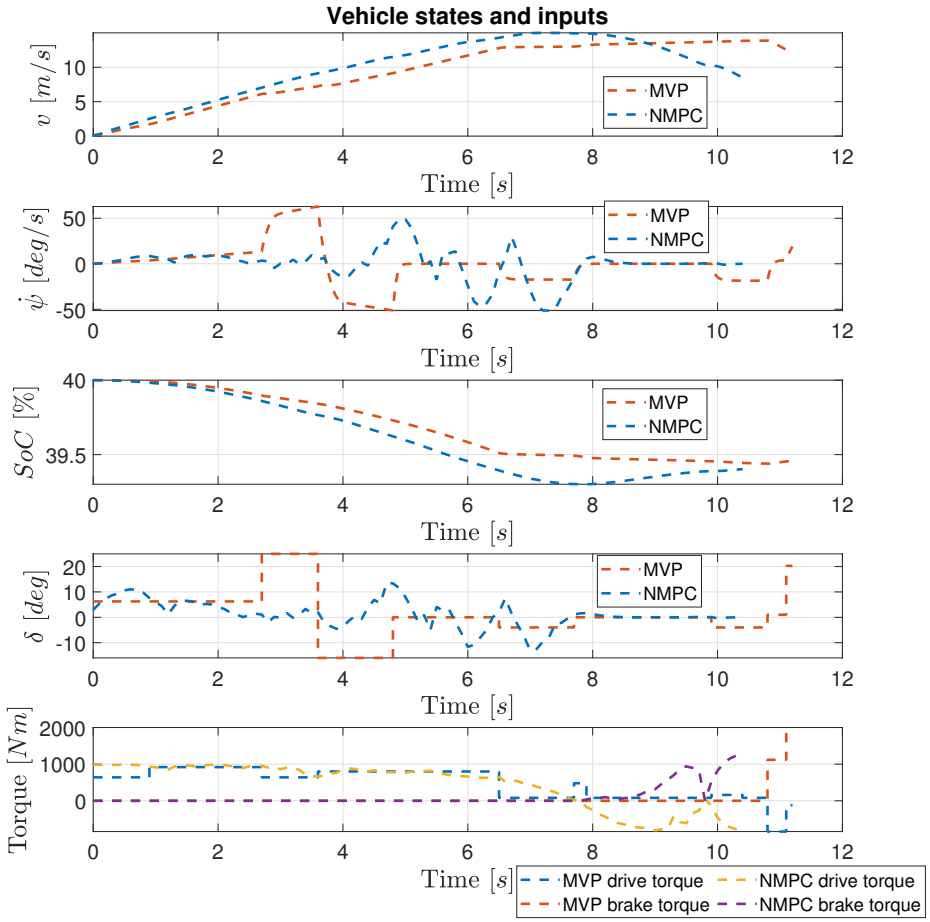


Figure 6.4: NMPC and MVP other states and inputs comparison in the second scenario in feedback simulation.

Collision with the obstacles was prevented for both methods. NMPC trajectory final time is better than MVP one. The MVP final SoC is slightly better than NMPC one. It cannot be definitely decided about better performance in this particular scenario. However, statistically it seems that NMPC gives slightly better results over various MVP runs for this particular case.

6.2 Comparison in general

6.2.1 Time demands

In the thesis, two-point boundary value problem (TPBVP) was identified as a main bottleneck of the original minimum-violation planning. TPBVP is needed to be solved for all neighboring nodes in every iteration when the new sample is added. Time demands for TPBVP solved by optimization tools are very high for systems with more than 3 states (and multiple outputs). Proposed version 1 (section 5.3.2) of the modified MVP with precomputed trajectories improves the computational time. However, the trajectories precomputation is possible only for slightly larger systems, mainly because of memory demands. For example, systems up to 5 states and 2 inputs (if possible, trajectory transformations are beneficial) are suitable for the precomputation. MVP version 2 (section 5.3.3) based on RRT was proposed for larger systems. The mean time demands for a tree with 1000 nodes and low complexity design model are stated in the following table 6.1. For version

MVP version / TPBVP solution	Computational time
Original MVP / optimization tool (CasADi)	5 min
Version 1 / trajectories precomputation	3 s
Version 2 / inputs integration (not a TPBVP)	5 s

Table 6.1: Time demands comparison for versions of MVP - dealing with TPBVP. Times given for a tree with 1000 nodes based on the low complexity design model on computer with parameters stated in appendix table B.5.

2, the computational time depends on used ordinary differential equation solver. TPBVP has linear time complexity in the number of nodes in the tree. It was observed in experiments that the TPBVP is the most demanding part of the MVP up to a certain number of nodes (order of ten thousand nodes), then the distances computation for nodes in neighborhood selection starts to be more demanding.

Used replanning method for feedback discards all nodes in the tree except the best path. Then a new whole tree has to be created again. Thus, it is very time consuming. The stated method for root reconnecting, while keeping the remaining nodes ((iii) in subsection 5.5.2), would give better performance. At the start, tree precomputation could be run and then in each iteration a certain number of nodes would be added (for example, 30 nodes per sample time 0.1 s).

The presented NMPC algorithm was not real time, too. The design models are quite large systems with 3 inputs. One NMPC iteration lasts about 0.3 s (sample time 0.1 s) on the design models. It depends on the number of SQP iterations and a lot of other parameters and constraints. However, the time could be reduced. The continuous models are given to NMPC, but NMPC can easily handle discrete time models. Procedures described in the NMPC one iteration overview (Alg. 1) are more demanding for continuous models. Mainly equations simulations by ordinary differential equation solver (ode45) are more demanding than simulations for discrete time models that are carried in a loop. Thus, a transformation of the continuous models to discrete time models can reduce the computational time.

6.2.2 Space searching comparison

As it was shown, NMPC as a solver requiring convexity needs heuristics for dealing with a nonconvex space. Other optimization procedure/tool is needed as a heuristics. Heuristics are usually specialized to certain problems. Therefore, the main algorithm (NMPC) is then suitable only for that problem. Heuristics Cost to go is used for the static obstacles in the thesis. The moving obstacle avoidance was handled only by the provisional solution as stuck at the local minima is possible. The designed Cost to go heuristics assigns costs to vertices. Then trajectory along the best vertices is searched. It assumes in advance that the obstacles in the space are not passable obstacles. However, the obstacle can be a grass strip or mud on the road. It can be preferred to go through that, in spite of that the grass has a higher cost than the surrounding road over performing a complicated maneuver. This is not solved for NMPC. Moreover, other constraints given by a surrounding environment are not solved by the NMPC. The traffic lights can be given as an example: "If the orange lights go with a velocity higher than a certain constant or stop." The example creates a nonconvex constraint. These and similar cases are not handled in NMPC. Other specialized heuristics will be needed for dealing with this type of issues. NMPC is not a sufficiently general algorithm to solve these issues.

On the other hand, all mentioned problems can be immediately formulated in MVP method and it will give proper results. For example, a grass obstacle can be added to the cost function with a suitable weight on the same priority level as the time. Then the optimization will weight between time and positions in grass. MVP has slightly worse results in the presented scenarios, but this is partly given by the algorithm considerable generality.

It was shown in the last scenario that MVP is not suitable for accurate

position tracking as it requires a lot of nodes in the graph and it is very time consuming (for large systems). Conversely, NMPC is an algorithm formed for these types of problems and therefore it gives better performance on these tasks.

Chapter 7

Conclusion and Future Work

Both algorithms model predictive control and minimum-violation planning were implemented and compared. The problem of planning in the environment with obstacles is defined in the chapter 2. In the next chapter 3, vehicle high-fidelity model and its components are adopted and described. Following chapter 4 solve the planning problem by nonlinear model predictive control (tool NMPC provided by Garrett motion). Design models suitable for control by NMPC are derived, the problem is solved via Cost to go heuristics and the results are stated. Minimum-violation planning method is discussed in the next chapter 5. Design models suitable for control by MVP are derived, modifications to MVP are proposed to be able to deal with large dynamic systems and results in open loop are stated. The possibilities of feedback for MVP are discussed and proposed. The chapter 6 compare these methods in test scenarios and in general.

There is a lot of possible improvements and unresolved issues that can be handled in the future work. For NMPC, there can be improvements in dealing with moving obstacles or improvements by time discretization of design models for computational time reduction. For MVP, the powerful part of the algorithm which can handle various formulas of linear temporal logic (that can be suitable eg. for traffic rules constraints definition) were discussed very briefly. Design model linearization in sampled states and using some of known methods for two-point boundary value problem in linear systems is for consideration as it could reduce the computational time. Only MVP based on rapidly-exploring random tree was discussed, other sampling-based algorithms could bring improvements (for example, dense sampling around the obstacles). Trajectory post-processing could solve the roughness of the MVP trajectory. In addition, the combination of both methods - raw trajectory generation by

MVP and trajectory smoothing and tracking by NMPC - is an interesting idea.

Appendix A

Bibliography

[1] Jon Louis Bentley. Multidimensional binary search trees used for associative searching. *Commun. ACM*, 18(9):509–517, September 1975. doi:10.1145/361002.361007.

[2] Francesco Borrelli, Alberto Bemporad, and Manfred Morari. *Predictive Control for Linear and Hybrid Systems*. URL: http://www.mpc.berkeley.edu/mpc-course-material.

[3] H Bouchareb, K Saqli, N M&apos, ; Sirdi, M Oudghiri, A Naamane, and N K M'sirdi. Electro-thermal coupled battery model : State of charge, core and surface temperatures estimation. Technical report. URL: https://hal.archives-ouvertes.fr/hal-02486440.

[4] Luis I Reyes Castro, Pratik Chaudhari, Jana Tůmová, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Incremental Sampling-based Algorithm for Minimum-violation Motion Planning. Technical report. arXiv:1305.1102v2.

[5] Denis Efremov. Unstable ground vehicles and artificial stability systems. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2018.

[6] T. D. Gillespie. *Fundamentals of Vehicle Dynamics*. Society of Automotive Engineers, Warrendale, PA, c1992.

[7] Benjamin Gutjahr, Lutz Gröll, and Moritz Werling. Lateral vehicle trajectory optimization using constrained linear time-varying mpc. *IEEE Transactions on Intelligent Transportation Systems*, 18(6):1586–1595, 2017. doi:10.1109/TITS.2016.2614705.

[8] Christian Götte, Martin Keller, Christoph Rösmann, Till Nattermann, Carsten Haß, Karl-Heinz Glander, Alois Seewald, and Torsten Bertram. A real-time capable model predictive approach to lateral vehicle guidance. In *2016 IEEE 19th International Conference on Intelligent Transportation Systems (ITSC)*, pages 1908–1913, 2016. doi:10.1109/ITSC.2016.7795865.

[9] Lukáš Haffner. *Real-time tire models for lateral vehicle state estimation.* Phd thesis, Technischen Universität Wien, Fakultät für Maschninenbau, Wien, 2008.

[10] Kerem Koprubasi. *Modeling and control of a hybrid-electric vehicle for drivability and fuel economy improvements.* PhD thesis, The Ohio State University, January 2008. URL: https://ui.adsabs.harvard.edu/abs/2008PhDT.....125K.

[11] Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng. Path planning for autonomous vehicles using model predictive control. In *2017 IEEE Intelligent Vehicles Symposium (IV)*, pages 174–179, 2017. doi:10.1109/IVS.2017.7995716.

[12] Marek Lászlo. Flight control solutions applied for improving vehicle dynamics. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2019.

[13] Douglas L. Milliken and William F. Milliken. *Race car vehicle dynamics.* SAE International, Warrendale, PA, c2003.

[14] Martin Mondek. Active torque vectoring systems for electric drive vehicles. Master's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2018.

[15] Kourosh Naderi, Joose Rajamäki, and Perttu Hämäläinen. RT-RRT*: A Real-Time Path Planning Algorithm Based On RRT*. URL: http://dx.doi.org/10.1145/2822013.2822036, doi:10.1145/2822013.2822036.

[16] Hans B. Pacejka. *Tire and Vehicle Dynamics.* SAE International, 2nd edition, 2005.

[17] Vasumathi Raman, Alexandre Donzé, Mehdi Maasoumy, Richard M Murray, Alberto Sangiovanni-Vincentelli, and Sanjit A Seshia. Model Predictive Control for Signal Temporal Logic Specifications. Technical report. arXiv:1703.09563v1.

[18] James B Rawlings, David Q Mayne, and Moritz M Diehl. *Model Predictive Control: Theory, Computation, and Design 2nd Edition.* URL: http://www.nobhillpublishing.com/mpc-paperback/index-mpc.html.

[19] Arthur Richards and Jonathan How. Mixed-integer Programming for Control. Technical report. URL: http://hohmann.mit.edu/milp/.

[20] Dieter Schramm, Manfred Hiller, and Roberto Bardini. *Vehicle dynamics*. Springer, New York, 2014.

[21] Jana Tumova, Gavin C. Hall, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Least-violating control strategy synthesis with safety rules. In *Proceedings of the 16th International Conference on Hybrid Systems: Computation and Control*, HSCC '13, page 1–10, New York, NY, USA, 2013. Association for Computing Machinery. doi:10.1145/2461328.2461330.

[22] Petr Turnovec. Vehicle Slip Ratio Control System for Torque Vectoring Functionality. Bachelor's thesis, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2019.

[23] Jana Tůmová, Luis I Reyes Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus. Minimum-violation LTL Planning with Conflicting Specifications. Technical report, 2013. arXiv:1303.3679v1.

[24] Vojtěch Vonásek. Motion planning algorithms. Autonomous robotics course, Czech Technical University in Prague, Faculty of Electrical Engineering, Praha, 2020.

[25] Tichakorn Wongpiromsarn, Konstantin Slutsky, Emilio Frazzoli, and Ufuk Topcu. Minimum-Violation Planning for Autonomous Systems: Theoretical and Practical Considerations. arXiv:2009.11954v1.

[26] Tichakorn Wongpiromsarn, Ufuk Topcu, and Richard M Murray. Receding Horizon Temporal Logic Planning. Technical report, 2010. URL: http://www.cds.caltech.edu/{~}murray/papers/wtm10-tac.html.

Appendix B

List of Abbreviations, Symbols and Parameters

Abbreviations

ABS	Anti-lock braking system
CL	Center line
CoG	Centre of Gravity
CS	Coordinate system
EM	Electric motor generator
ESP	Electronic stability program
(F)LTL	(Finite) linear temporal logic
LQR	Linear-quadratic regulator
MIP	Mixed-integer programming
MPC	Model predictive control
MVP	Minimum-violation planning
NMPC	Nonlinear model predictive control (tool provided by Garrett motion)
PID	Proportional–Integral–Derivative Controller
PRM	Probabilistic roadmap
RRG	Rapidly-exploring random graph
RRT	Rapidly-exploring random tree
SQP	Sequential quadratic programming
TC	Traction control
TPBVP	Two-point boundary value problem
TV	Torque vectoring

Symbols

T_{req}	Electric motor - generator requested torque
ω_{in}	Electric motor - generator shaft angular speed
T_{int}	Electric motor - generator torque
ω_{em}	Electric motor - generator angular speed
T_{em}	Electric motor - generator out torque
P_{em}	Electric motor - generator power
η	Electric motor - generator efficiency
P_{trm}	Battery terminal power
SoC	Battery state of charge
v_1	Battery circuit voltage
v_2	Battery circuit voltage
i	Battery circuit current
V_{trm}	Battery terminal voltage
V_{oc}	Battery open circuit voltage
R_0	Battery internal resistance
δ	Wheel steering angle
T_ω	Wheel drive torque
$T_{\omega,br}$	Wheel brake drive torque
v_x	Wheel brake translation velocity in longitudinal direction
v_y	Wheel brake translation velocity in lateral direction
F_z	Wheel vertical load
ω	Wheel angular velocity
λ	Tire slip ratio
α	Tire slip angle
F_{aero}	Vehicle aerodynamics drag force
F_{slope}	Vehicle slope force
v	Vehicle velocity
β	Vehicle side slip angle
$\dot{\psi}$	Vehicle yaw rate
ψ	Vehicle yaw angle (heading)
e	Vehicle east position
n	Vehicle north position
F_x	Longitudinal force
F_y	Lateral force
α_{inc}	Road inclination (slope) angle
M_z	Moment around vehicle CoG

Subscripts

f	Front - wheel, tire, electric motor, ...
r	Rear - wheel, tire, electric motor, ...
x	Longitudinal direction - wheel, vehicle, ...
y	Lateral direction - wheel, vehicle, ...

Parameters

Description	Notation	Value	Unit
Torque time constant	t_T	0.05	s
Angular speed filter constant	t_ω	0.5	s
Rotor moment of inertia	J_{em}	0.01	kgm^2
EM minimal torque	$T_{int,min}$	-100	Nm
EM maximal torque	$T_{int,max}$	100	Nm
Gear ratio	r_ω	10	-

Table B.1: Vehicle electric motor - generator parameters.

Description	Notation	Value	Unit
Battery capacity	C	27.2	Ah
Resistance in R_1C_1 pair	R_1	0.08	Ω
Resistance in R_2C_2 pair	R_2	0.04	Ω
Capacitor in R_1C_1 pair	C_1	0.8	F
Capacitor in R_2C_2 pair	C_2	0.4	F

Table B.2: Vehicle battery parameters.

Description	Notation	Value	Unit
Tire radius	R	0.3	m
Front tire moment of inertia	J	1.0	kgm^2
Longitudinal stiffness factor	B_x	10	-
Longitudinal shape factor	C_x	2.05	-
Longitudinal peak factor	D_x	1.0	-
Longitudinal curvature factor	E_x	0.6	-
Lateral stiffness factor	B_y	5.73	-
Lateral shape factor	C_y	2.0	-
Lateral peak factor	D_y	1.0	-
Lateral curvature factor	E_y	0.6	-

Table B.3: Vehicle tire parameters.

Description	Notation	Value	Unit
Vehicle mass	m	1000	kg
Vehicle vertical axis moment of inertia	I_z	1000	kg
Longitudinal distance of front axle from center of gravity	l_f	1.3	m
Longitudinal distance of rear axle from center of gravity	l_r	1.7	m
Gravitational acceleration	g	9.81	ms^{-2}
Aerodynamic reference area	A_f	2.23 m	m^2
Air density	ρ	1.225	kgm^3
Aero dynamic drag coefficient	c_d	0.304	-

Table B.4: Vehicle parameters.

Description	Parameter
System model	HP ProBook 640 G4
Operating system	Microsoft Windows 10 Pro
System Type	x64-based PC
Processor	Intel(R) Core(TM) i5-8350U CPU @ 1.70 GHz, 1896 Mhz, 4 Cores, 8 Logical Processors
RAM	8.00 GB

Table B.5: Computer parameters.

Appendix C

Content of Enclosed CD

- The thesis in pdf format.
- Source files - majority of the files is not executable because of NMPC tool licensing by Garrett motion and because the high-fidelity models of vehicle components were implemented in Garrett motion MATLAB class suitable for it, which cannot be published. However, the included configuration files are quite easily understandable and are enclosed for illustration.
- Simulation videos for the presented scenarios and one new scenario.