Master Thesis

Czech Technical University in Prague

F3

Faculty of Electrical Engineering
Department of Control Engineering

Autonomous vehicle trajectory tracking algorithms

Bc. Jan Švancar

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.
January 2023

Acknowledgements

I would like to express my deepest gratitude to my supervisor doc. Ing. Tomáš Haniš, Ph.D. for excellent guidance during my work on this thesis. Special thanks go to my family and fiancé for their endless support. Last but not least, another special thanks goes to my fellow colleagues Ing. Marek Boháč, Ing. Adam Konopiský, Ing. Tomáš Twardzik and Ing. Tomaš Veselý for their emotional support during sad, long and exhaustive debug and repair sessions.
This work was supported by TRACE.

Declaration

I declare that I have prepared the submitted work independently and I have listed all the used literature.

In Prague, 10. January 2023

Abstract

This work is looking into the problem of autonomous vehicle path tracking. The purpose of this work is to design and implement path tracking algorithm that could reflect the capabilities of the over-actuated vehicle platform and test this algorithm against the baseline path-tracking algorithm.
The vehicle platform that was used in this work is an RC vehicle modified so that it can independently turn every wheel. The platform has also got distributed computational power in a form of several units which are part of the ROS2 network.
The baseline algorithm used for purpose of this work was the Stanley Control Law, which was designed at Stanford University for purpose of controlling the autonomous vehicle in the DARPA Challenge.
The other algorithm designed was MPC based algorithm that could reflect that the vehicle platform can independently steer all wheels.
The algorithms deployed on the vehicle platform were implemented a the ROS2 nodes to be able to communicate with the rest of the vehicle.
The path-tracking algorithms were tested in the real environment within the park of the CTU FEE on Charles Square.

Keywords: MPC, Stanley Control Law, autonomous vehicle, path tracking, over-actuated platform, ROS2

Supervisor: doc. Ing. Tomáš Haniš, Ph.D.
Department of Control Engineering,
CTU in Prague - FEE
Karlovo Naměstí 13,
Praha 2

Abstrakt

Tato se práce se zabývá problémem vedení autonomního vozidla po referenční cestě. Cílém této práce bylo vytvořit a implementovat algoritmus pro sledování cestz, který by byl schopný brát v úvahu možnosti přeauktuované platformy a otestovat tento algoritmus ve srovnání se základním algoritmem pro sledování cesty.
Testovací platforma použitá v této práci je RC vozidlo modifikováno tak, aby bylo schopno nezávisle natáčet všechnz kola. Platforma má také distribuovanou výpočetní sílu v podobě několika výpočetních jednotek, které jsou součástí sítě ROS2.
Základní algoritmus použitý v této práci je tak zvaný Stanley Control Law, který byl navržen na Stanfordské Univerzitě, pro potřeby sledování cesty autonomním vozidlem při DARPA Challenge.
Dalším použitým algoritmem je algoritmus založený na MPC, který je schopný využít nezávislého natáčení všech kol, čehož je platforma schopna.
Algoritmy nasazené na platformě bzli implementovány jako uzly sítě ROS2, aby byli schopné komunikovat se ybztkem platformy.
Algoritmy byli testovány v prostředí parku uvnitř komplexu budov ČVUT na Karlově Náměstí.

Klíčová slova: MPC, Stanley Control Law, autonomní vozidlo, sledování cesty, přeaktuaovaná platforma, ROS2

Překlad názvu: Algoritmy vedení po trati pro autonomní vozidlo

Contents

Figures

Tables

Chapter 1

Introduction

1.1 Motivation

This work was highly motivated by papers published at the ITCS conference [VCH+21] and [CHK+21]. The mentioned papers were aimed to examine a way to estimate and predict parameters of the driving surface using image data from cameras placed somewhere on the vehicle. Image data can then provide prior information about what surface is ahead of the vehicle so the assistance systems or algorithms deployed on a vehicle can be adjusted to have better performance on the given driving surface. An example of this behavior could be seen for example in [BFP+20], where road friction properties are provided to the braking control algorithm in order to adjust vehicle response. The problem of adjusting vehicle parameters based on a prediction of surface properties can be then reformulated into a problem of following the path with optimal driving parameters. That means that based on image data one can estimate surface properties ahead of the vehicle and then plan a path that will lead the vehicle to the positions with the optimal driving surface. This could be especially useful when for example vehicle has to avoid frozen-over places on the road.
Another motivation for this work was to examine the possibilities of a relatively new concept of over actuated vehicles. The over actuation of the vehicle can lie in having a vehicle with all wheels steered or powered independently. This concept can be then used to develop control or assist algorithms with better performance in certain situations.

1.2 Problem Formulation

The fundamental problem here is to develop a pipeline of algorithms, which will allow the vehicle to classify surface ahead and independently on the human operator lead vehicle through a surface with optimal driving properties. This problem can be further divided into three subproblems namely prediction of surface properties from image data, path planning with respect to predicted data and path tracking.

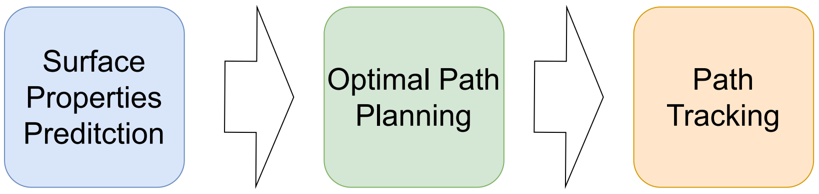


Figure 1.1: Full pipeline

The surface properties prediction algorithm is described in [Kon22]. This algorithm uses an artificial neural network to process images from a camera placed on the vehicle in order to obtain an estimation of surface properties.

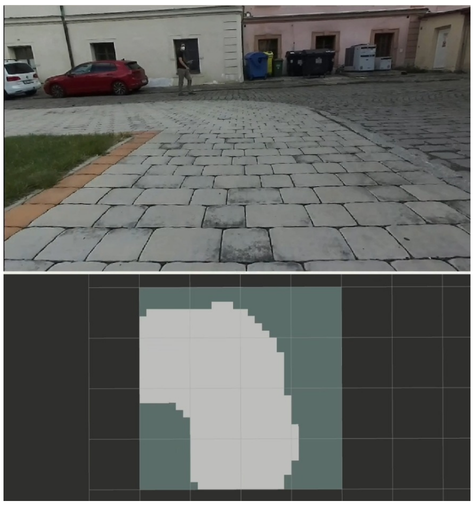


Figure 1.2: Example of the image processing [Kon22]

The second part of the pipeline is a path planning algorithm described in [Boh22]. This algorithm uses data from the image processing part to plan the optimal path through the environment. The path planned by this algorithm chooses the best available surface according to several conditions described in related work.

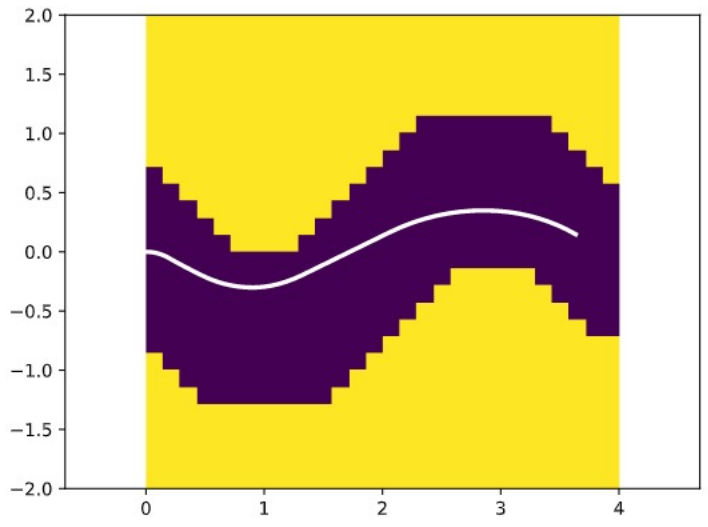


Figure 1.3: Path generated by planning algorithm [Boh22]

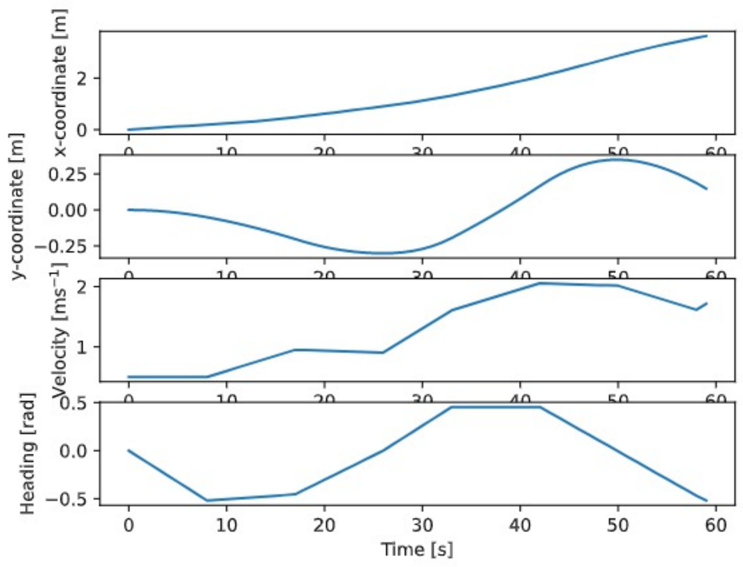


Figure 1.4: Planned development of states [Boh22]

The last part of the pipeline is the path-tracking part. This algorithm uses the path obtained from the path planning part to safely navigate the vehicle along it. The development of this part of the pipeline is the main focus of this work.

1.3 State of the Art

1.3.1 Over-actuated vehicles

The over-actuated vehicle is the vehicle that has got more forms of actuation than it is necessary to control it. The over-actuation in our platform lies in the independent steering of all wheels.
Another example of the over-actuated vehicle might be the vehicle platform used in [CW14]. The vehicle used there has got four inwheel BLDC motors and front wheel steering.

1.3.2 Path planning

There are many approaches to the path planning problem. The simplest approach is to use the algorithm from the family of graph-based algorithms. This family groups algorithms like A*, D*, the Dijkstra algorithm, breadth-first search and many others. One example of the usage of graph-based algorithms can be seen in [SNK19]. In this work the Hybrid A* algorithm is used to plan the optimal path to the parking spot. However, the drawback of the graph search algorithm is that the quality of output of these algorithms strongly depends on the resolution of the searched space and it is quite difficult to incorporate the constraint on vehicle dynamics or kinematics into the algorithm.
Another group of path-planning algorithms is based on sampling methods. These algorithms are based on a random sampling of the given state space. For example, in [KFT$^+$08] the RRT-based algorithm is used to plan a path in the urban environment. The algorithm selects a starting point (measurements in the given time), take a random sample of the input to the mathematical model and apply the input to the mathematical model. The mathematical model is then "simulated" with the applied inputs and the states, where the simulation ends, are marked as a node of the algorithm. The node and the state trajectory are then compared with a feasibility map to check if there are any possible collisions. After the collision check the node is added to the search tree.
The MVP (minimum violation planning) framework [RCCT$^+$13, TRCK$^+$13] based on RRt* also belongs to sample-based planning methods. The advantage of the MVP framework is that it can handle logical conditions and also it can handle multiple rules of different importance.
There are also optimization-based methods for path planning. These on

defining the path planning problem as the optimization problem. A notable member of this family is MPC based path planning algorithm described in [LLVT17]. The drawback of the MPC-based planning algorithm is that the algorithm needs the convex optimization problem to find the globally optimal solution and not be stuck in the local optimum. The algorithm then needs some sort of heuristic for example a cost-to-go heuristic from [RH05].

1.3.3 Path tracking

There are many path-tracking algorithms that can be used to navigate autonomous vehicles and robots in general.
According to [BSA+17], path-tracking algorithms can be divided into several categories, geometric algorithms, kinematic algorithms and dynamic algorithms.
Geometric algorithms take into consideration only vehicle position and orientation and its geometry. These algorithms are the simplest ones because they don't reflect any internal or external forces affecting the vehicle or the velocities of the vehicle.
Kinematic algorithms are designed using a kinematic vehicle model. This means that these algorithms take into consideration also the velocity and angular velocity of the vehicle. However, these algorithms still don't consider forces affecting the vehicle.
The last group of algorithms is the group of dynamic control algorithms. These algorithms are designed using dynamic vehicle models and are capable of considering also forces affecting the controlled vehicle.

Geometric Control Algorithms

- Carrot Following Algorithm
  This algorithm uses so-called lookahead point $s = \begin{pmatrix} x_s & y_s \end{pmatrix}^T$ which is a point on a path in a given distance from the vehicle reference point. Then let vector $v = \begin{pmatrix} x & y \end{pmatrix}$ be the position of the vehicle, ψ be its heading and V vector describing vehicle orientation. There can be introduced vector $d = s - v$ and the angle between vector d and vector V can be denoted as ψ_d which is also the tracking error.

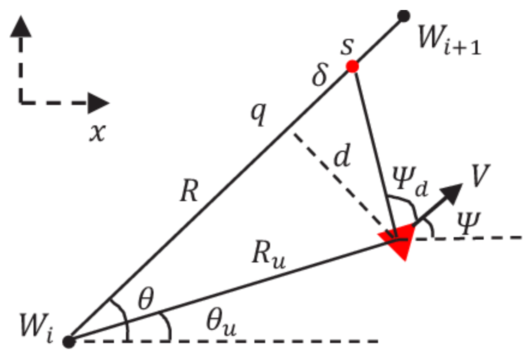


Figure 1.5: Ilustration of carrot following algorithm functionality from [TYKAM15]

The cornering ability of this algorithm strongly depends on the structure of the controlled robot and also on the choice of the distance of the lookahead point. If the distance of the lookahead point is too long the robot will suffer from large deviation from the given path and can end up in dangerous situations. On the other hand, if the lookahead distance is too short then the robot doesn't have to be able to track curves with a smaller radius and is more likely to overshoot.
This algorithm is mainly used for two-wheeled circular-shaped robots because these robots have much simpler steering geometry than vehicles.

- Pure Pursuit Algorithm
  Pure pursuit is an algorithm used for front-wheel steering vehicles. Its reference point is situated in the center of the rear axle of the vehicle. This algorithm also uses the concept of the look-ahead point (see section 1.3.3).
  This algorithm uses the center point of the rear axle as its reference point. The algorithm then creates a circular arc that will lead the reference to the look-ahead point. The control action (steering angle of the front axle wheels) is then calculated based on the circular arc using the equation 1.1.

$$\delta_F = \arctan(\frac{2L\sin(\alpha)}{l_d}), \tag{1.1}$$

where δ_F is the steering angle of the front axle, L is the distance between the front and rear axle, α is the difference between the vehicle heading and heading of the line segment between the reference point and look-ahead point and d is a distance of the look-ahead point from the reference point.

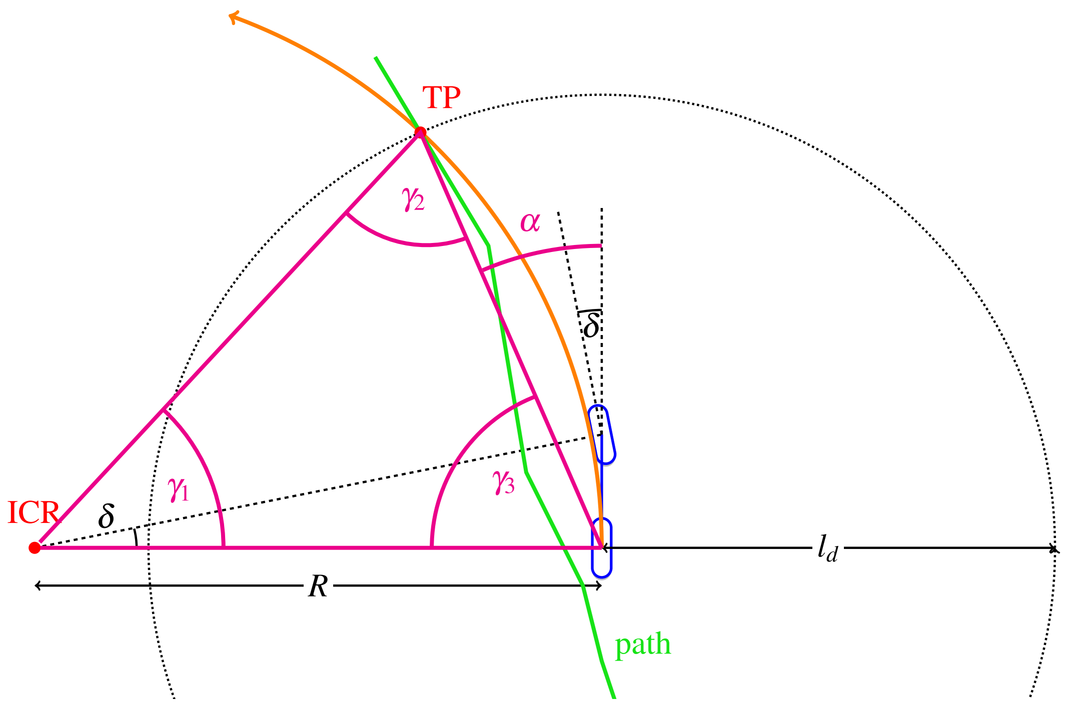


Figure 1.6: Illustration of pure pursuit algorithm [TS]

- Stanley Control Law
  Stanley Control Law is a path-tracking algorithm that was invented for purpose of controlling autonomous the vehicle called Stanley through the DARPA challenge. This algorithm was first introduced in article [HTMT07]
  This algorithm uses two error metrics to track the desired path. The first error metric is the so-called cross-track error e_c, which describes the signed perpendicular distance of the vehicle's front axle from the reference path. The second error metric is heading error $e_\psi = \psi_{ref} - \psi$, which describes the deviation of vehicle body heading ψ from reference heading ψ_{ref}. Base Stanley Control Law can then be described by equation 1.2.

$$\delta(t) = (\psi_{ref}(t) - \psi(t)) + \arctan(\frac{e_c(t)}{k_s + v(t)}) \tag{1.2}$$

The Stanley Control Law used in the DARPA challenge has two additional features. The first additional feature is a yaw rate damper. This damper can be described by equation 1.3.

$$k_{d,yaw}(r_{meas} - r_{traj}), \tag{1.3}$$

where $k_{d,yaw}$ is a tuning parameter, r_{meas} is the vehicle yaw rate and r_{traj} is an estimated yaw rate vehicle has to achieve to precisely track the reference path. The second additional feature is the steering damper which can be described by equation 1.4.

$$k_{d,steer}(\delta_{meas}(i) - \delta_{meas}(i+1)) \tag{1.4}$$

where $k_{d,steer}$ is a tuning parameter, δ_{meas} is steering angle on the servos in given moment, i is the index of the measurement one period earlier. This additional feature should damp the response of the steering mechanism, servos in our case.

Kinematic Control Algorithms

Kinematic algorithms are algorithms designed with the use of a kinematic model of the controlled vehicle.
One example of the MPC-based tracking algorithm comes from [PSLM17]. This MPC-based algorithm uses a discrete-time kinematic model of the four-wheel steered vehicle as a predictor for the MPC algorithm. The discrete-time model can be described by the equations below.

$$X_{i+1} = X_i + T_s v_i \cos(\Psi_i + \beta_i), \tag{1.5}$$
$$Y_{i+1} = Y_i + T_s v_i \sin(\Psi_i + \beta_i), \tag{1.6}$$
$$\Psi_{i+1} = \Psi_i + T_s v_i \kappa_i, \tag{1.7}$$

where X is the position on the x-axis, Y is the position on the y-axis, Ψ is the vehicle body heading, β is side-slip angle and is taken as a first input and κ is a curvature of the curve vehicle has to do and is taken as a second input. This model is then linearized using the reference points on the path and used as the predictor for the MPC framework.
This particular formulation gives optimal side-slip angle β^* and optimal curvature κ^*. These are not exactly the control inputs for the vehicle so κ^* and β^* have to be recalculated into front wheel steering angle δ_F and rear wheel steering angle δ_R using geometric properties of Ackerman steering geometry. Another approach proposed in [TYZ$^+$20] is to use MPC in cascade with PID controller on a yaw rate of the vehicle. The MPC algorithm is designed to track the position and heading of the vehicle and output the optimal yaw rate. The yaw rate from the MPC is then used as a reference yaw rate for the PID controller. This cascade connection according to the authors will result in a significant reduction of tracking errors.

Dynamic Control Algorithms

Dynamic control algorithms are algorithms that are designed using a dynamic model of the vehicle. They can reflect for example forces affecting the vehicle's wheels or the vehicle itself.
One of the examples of the dynamic control algorithm comes from [YGP$^+$19].

There was designed a robust MPC controller based on a steady-state error model.
There is also a possibility to use NMPC (Non-linear model predictive control) introduced in [LLZ$^+$19]. This work uses NMPC to track the desired path. The NMPC then outputs the optimal front wheel steering angle δ_F and also a requirement on the yaw moment of the vehicle M_z. The requirement on the yaw moment of the vehicle is then handed over to the distribution logic, which will select the appropriate braking torques on the wheels.

Adaptive Control Algorithms

The control algorithm proposed in [SZC$^+$20] uses a combination of pure pursuit algorithm and PID controller. These algorithms are combined using reinforcement learning.
Another approach introduced in [LJKK19] is to LQR (linear quadratic regulator) controller with Kalman Observer. This control technique is enhanced with lookahead measurements. That means the measurements are also taken some distance in front of the vehicle. This distance is also adaptively changed.
The work [PN14] is solving the vehicle overtaking problem. The proposed solution is to design a nonlinear controller based on the relative kinematics of the two vehicles. The unknown velocity of the overtaken vehicle is then estimated based on error metrics. The estimated velocity of the overtaken vehicle is then used to adaptively tune the nonlinear control law.

Chapter 2

Vehicle Platform

2.1 Introduction

The vehicle platform is built on the commercially available RC car Losi Desert Buggy. The RC car was further modified to meet up requirements for developing advanced algorithms.

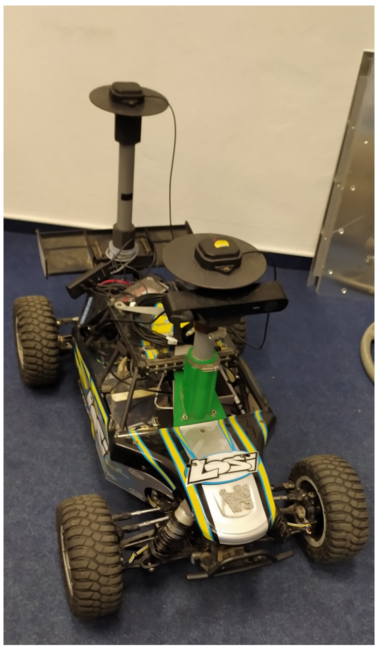


Figure 2.1: Vehicle platform

2.2 Hardware Configuration

2.2.1 Drivetrain

The vehicle platform is powered by one BLDC engine with motor controller. The engine is then connected through the shaft and differential to the rear wheel axle. The vehicle has only the rear wheels powered.
The engine is controlled by changing the width of the PWM signal applied to the motor controller. This change is done using the ROS2 node directly controlling hardware actions which then communicate with Navio using the serial interface.
Control signal $u_D \in [-100, 100]$ is the signed percentual value of allowed PWM width, where $u_D = 100\%$ is the maximal allowed value of PWM width and $u_D = -100\%$ is the minimal allowed PWM width. The value u_D is then recalculated to the PWM width and Navio then set the appropriate PWM output to this width.

2.2.2 Steering mechanism

The vehicle platform is equipped with four standalone servo motors, which independently control every wheel's steering angle. This steering configuration allows the development of advanced control algorithms for over-actuated vehicles.

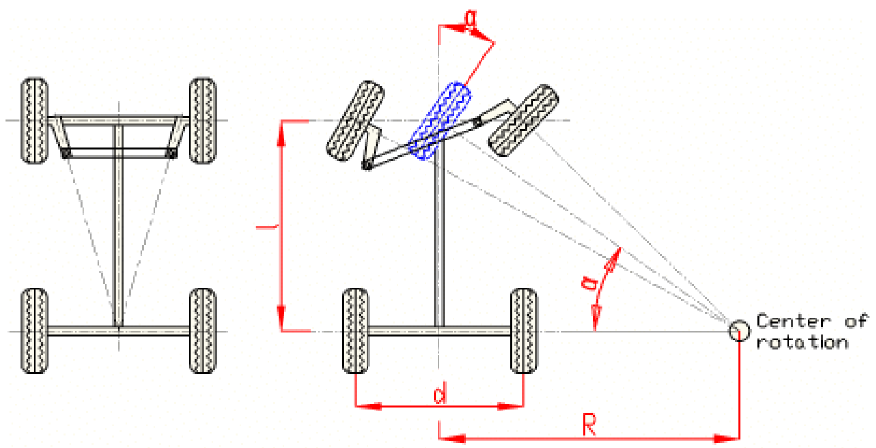


Figure 2.2: Ackermann steering principle [HRK10]

However, in this work, the single-track vehicle approximation is used. That means that the wheels which are on the same axle are moving with each other according to the rules of the Ackermann steering mechanism. The Ackermann steering mechanism causes the wheels on the same axle to turn at different angles so that the circular trajectory of both wheels has the same center. This mechanism helps to prevent additional slips of wheels during turn maneuvers. The Ackermann steering mechanism is implemented on both the front and the rear axles. The vehicle platform has this mechanism implemented in a software manner opposite to commercially produced cars that have this mechanism implemented mechanically.

2.2.3 Computational Units

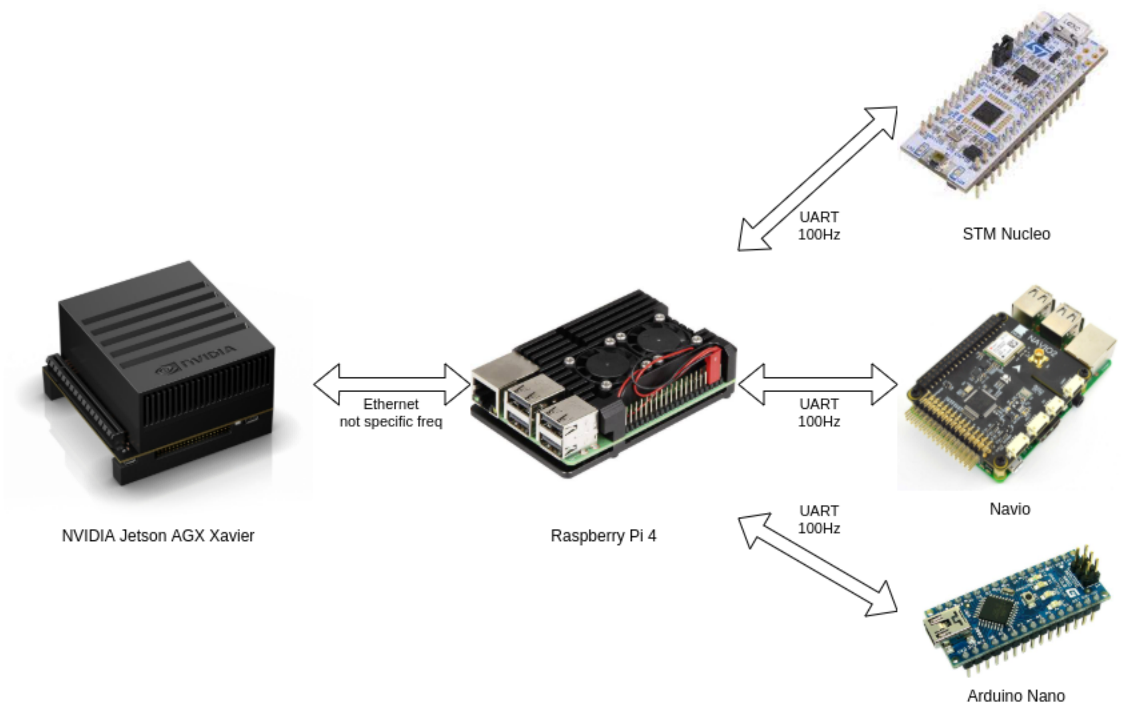


Figure 2.3: Computational units connections [BTKS]

the computational power of the vehicle platform is distributed into several computational units.

Computational units list.

- NVIDIA Jetson AGX Xavier
- Raspberry Pi 3B with Navio HAT (further in text only referred to as Navio)
- Raspberry Pi 4B (further in text only referred to as Central Raspberry)
- Microcontrollers
  - Arduino Nano
  - STM Nucleo

NVIDIA Jetson AGX Xavier serves as a graphic accelerator and is used to compute high-level tasks like forward propagation in an artificial neural network.
Navio is used to measure certain data and furthermore, Navio is equipped with PWM output.

2.2.4 Low-level Hardware Modified

Low-level microcontrollers and the hardware circuits were at first soldered on universal PCB (illustration can be seen in Fig. 2.4) with external wire connections soldered directly to the board. This design was prone to damage mechanical damage. The external wires often break in the soldered point. The circuits were almost impossible to repair when something got broken because the wires were directly soldered to the circuit and the circuit then couldn't be extracted from the vehicle platform. There was a need to find a solution to these problems.

The first change in the platform was the design of the printed circuit boards (further used as PCB) for the microcontrollers and the safety circuit. The design of the PCBs was done using open-source software named KiCAD (`https://www.kicad.org/`). This change then means that connections between elements in circuits are no longer connected using wires but they are connected by printed copper paths. This fact also contributes to the general resistance of the circuits to the effects of the environment.
The designed PCBs were equipped with connectors to solve the problem of extracting the PCBs from the platform. The connectors used in this particular case are 9-pin DSub Cannon connectors (`https://cz.mouser.com/ProductDetail/571-5747840-6`) and 15-pin DSub Cannon connectors (`https://cz.mouser.com/ProductDetail/649-10090926-P156VLF`).

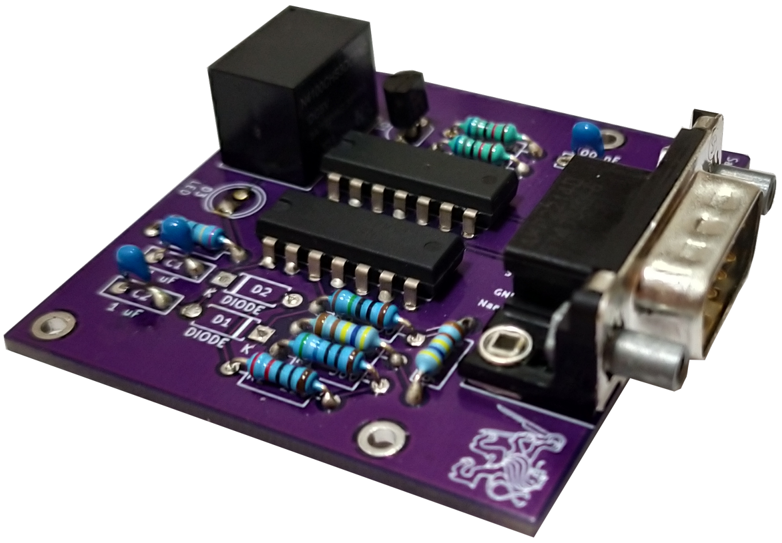


Figure 2.4: Manufactured and assembled PCB of the safety circuit

All the single wires were also changed for jacketed cables to reduce the effect of the electrical disturbance on the signals transferred through cables. All the PCB designs and final products will be shown in the Appendix of this work.

2.3 Communication Network

Communication between computational units is held by two kinds of interfaces. The first interface used is the Ethernet connection this connection is used by the ROS2 layer, which is establishing the connection between Jetson Xavier and central raspberry. The second interface is UART. UART is used to collect data from microcontrollers (Nano, Nucleo) and Navio and also to send control signals back to Navio, which then set appropriate values for PWM outputs.

2.3.1 ROS2 Network

ROS2 network creates an application layer above the Ethernet connection of the Raspberry Pi 4B and Jetson Xavier. The ROS2 network provides some basic synchronization tools and communication tools.
ROS2 network in this particular case is used mainly for its communication tools. It allows us to asynchronously send and receive messages between Raspberry Pi and Jetson Xavier.

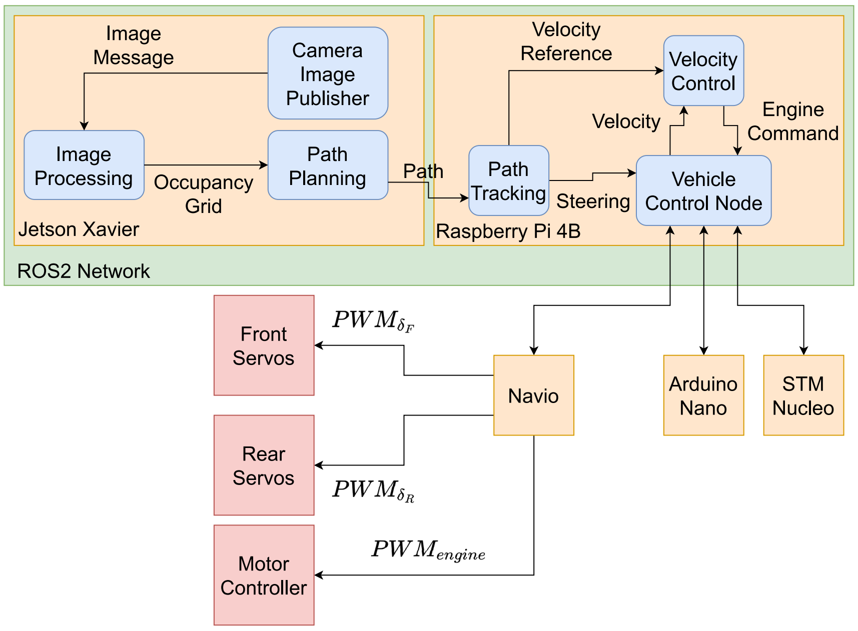


Figure 2.5: Communication architecture

In this case, the ROS2 network transfers image data from the ZED camera, transformations between coordinate frames, path data, the control commands from the controllers and also other vital measurements.

2.3.2 Serial Connections

The serial connections are used to transfer the data from the low-level hardware to the Raspberry Pi 4B. The data are then processed by the ROS2 node deployed on the Raspberry Pi 4B and propagated to the ROS2 network in a form of messages. The serial connection between the Raspberry Pi 4B and Navio is also used to transfer direct commands for servos and a motor controller to the Navio.

Specifications of serial interfaces.

- Raspberry Pi 4B and STM Nucleo
  - Data transfer rate: 100 Hz
  - Baud rate: 115200 Bd
  - Data type: *int16*

  - Data length: 8
  - Number of bytes: 16 bytes

- Raspberry Pi 4B and Arduino Nano
  - Data transfer rate: 100 Hz
  - Baud rate: 230400 Bd
  - Data type: *float*
  - Data length: 8
  - Number of bytes: 32 bytes
- Raspberry Pi 4B and Navio
  - Data transfer rate: 100 Hz
  - Baud rate: 460800 Bd
  - Data type: *float*
  - Data length: 70
  - Number of bytes: 280 bytes

2.4 Measurements

2.4.1 Position measurement

The measurement of position data is done using a dual GPS on the vehicle and a base station GPS, which is positioned in a certain spot with a known position. The dual GPS setup is giving a precise measurement of the heading of the vehicle body. Base station GPS is used to compensate for an error, which arises from GPS signal passaging magnetosphere.
Positional data are measured with sample frequency $f_{GPS} = 10\text{Hz}$.

2.4.2 Heading measurement

There are two different headings measured. The first measured heading is the body heading ψ_b of the vehicle which describes the orientation of the vehicle body in the plane. This heading is measured by the dual GPS receivers which

have their antennas situated on the opposite sides of the vehicle.
The second measured heading is the travel heading ψ_t which describes the orientation of the velocity vector of the vehicle. This measurement is done using measured velocities in NED coordinate frame and then calculating the travel heading ψ_t using a formula 2.1.

$$\psi_t = \text{atan2}(v_E, v_N), \tag{2.1}$$

where v_E is the velocity to the east and v_N is the velocity to the north.

2.4.3 IMU units

The Navio unit has two inbuilt IMU units namely MPU9250 (`https://invensense.tdk.com/wp-content/uploads/2015/02/PS-MPU-9250A-01-v1.1.pdf`) and LSM9DS1 (`https://cdn.sparkfun.com/assets/learn_tutorials/3/7/3/LSM9DS1_Datasheet.pdf`). These units have magnetometers, accelerometers and gyroscopes on all three axes.

2.4.4 Velocity measurement

The vehicle platform has Hall effect sensors with magnetic rings mounted on the half-shaft of each wheel. These sensors are used to measure the frequency of wheel revolutions in revolutions per minute.
These measurements are collected by STM Nucleo mentioned in section 2.2.3. The microcontroller then resends these data via the UART interface to the Navio unit with frequency $f_{STM} = 100\,\text{Hz}$. These data are further processed in the Navio unit.
The Hall effect sensors on the front wheels are also used to calculate the tangential velocity of the wheels. Only the frequency of the front wheels is included in the calculation of velocity because front wheels aren't affected by the motor and thus we can assume that front wheels have zero slip in the longitudinal direction. Calculation of the front wheels' tangential velocity v_f is done using the equation

$$v_f = (\frac{f_{FL} + f_{FR}}{2}) \cdot \frac{1}{60} \cdot 2 \cdot \pi \cdot r, \tag{2.2}$$

where f_{FL} and f_{FR} are revolutions per minute of the front left and front right wheel and r is the radius of the vehicle wheel in meters.
Another velocity measurement then comes directly from GPS which provides a ground speed of the vehicle. This measurement is updated with the frequency of 10 Hz.

2.4.5 Side-slip angle measurement

Measurement of side-slip angle β is done using vehicle both vehicle headings mentioned in section 2.4.2. The side-slip angle β is then calculated using a formula 2.3.

$$\beta = \psi_t - \psi_b, \tag{2.3}$$

where ψ_t is the travel heading of the vehicle and ψ_b is the body heading of the vehicle.

2.5 Coordinate frames and transformations

2.5.1 Global coordinate systems

There are used two global coordinate systems. The first used coordinate system is the so-called Eart-Centered, Eart-Fixed (ECEF) coordinate system. This system has the origin in the center of gravity of Earth, Z-axis is directed towards the north pole, X-axis is directed towards the intersection Prime Meridian (zero longitude) and Equator (zero latitude) and Y-axis is perpendicular to both Z- and X-axis.

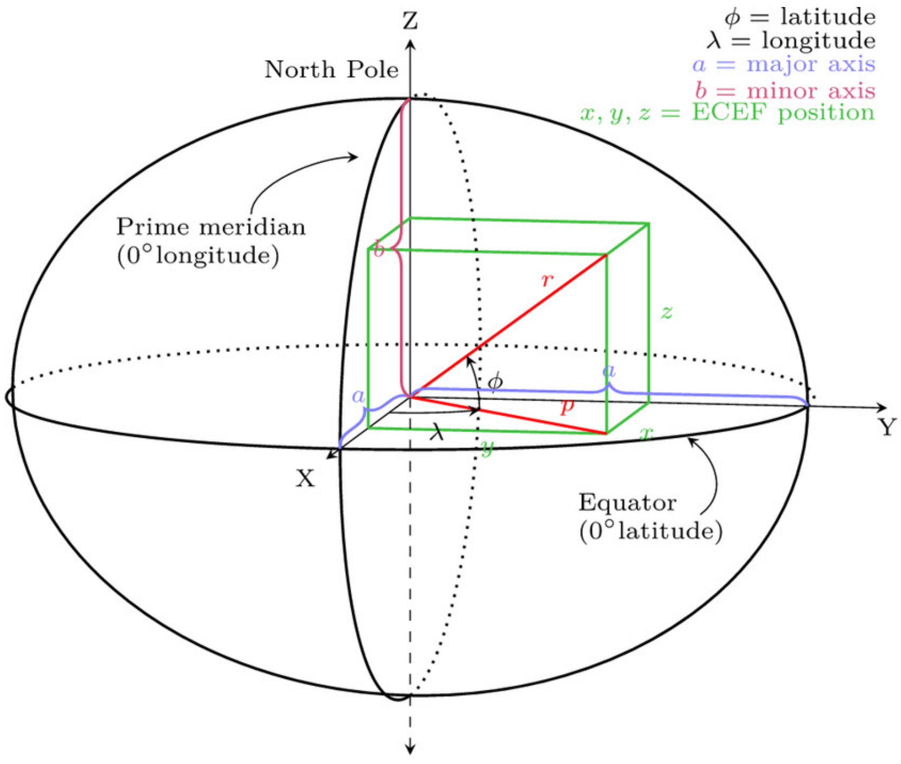


Figure 2.6: ECEF coordinate system [Şe17]

The second global coordinate system is the approximation of the North-East-Down coordinate system using a local tangential plane. Approximation of the global coordinate system by using a local tangential plane is done by the creation of a plane, which touches Earth in certain ECEF coordinates. These ECEF coordinates are then the origin point of the approximation of the NED coordinate system.

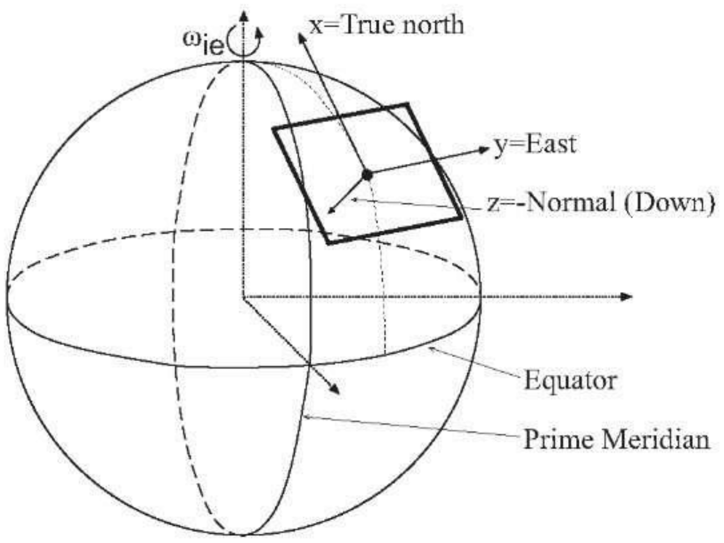


Figure 2.7: Local tangential plane NED coordinate system [MB16]

Coordinates in the ECEF coordinate system can be transformed into NED coordinates in the local tangent plane using the following equation

$$\vec{p}_{NED} = \mathbf{R}^T(\vec{p}_{ECEF} - \vec{p}_{ref}), \tag{2.4}$$

where $\vec{p}_{NED}$ is a vector of coordinates in the NED coordinate system, $\vec{p}_{ECEF}$ is a vector of coordinates in the ECEF coordinate system, $\vec{p}_{ref}$ is a vector of coordinates of the origin of the NED coordinate system in ECEF coordinate system and $\mathbf{R}$ is a rotation matrix. Rotation matrix $\mathbf{m}$ can be defined as

$$\mathbf{R} = \begin{pmatrix} -\sin(\phi)\cos(\lambda) & -\sin(\lambda) & -\cos(\phi)\cos(\lambda) \\ -\sin(\phi)\sin(\lambda) & \cos(\lambda) & -\cos(\phi)\sin(\lambda) \\ \cos(\phi) & 0 & -\sin(\phi) \end{pmatrix}, \tag{2.5}$$

where ϕ is latitude and λ is longitude. Similarly, this conversion can be also used to convert the NED coordinates back to the ECEF coordinates.
The nature of this approximation implies that this approximated NED coordinate frame holds only in the local neighborhood of reference point $\vec{p}_{ref}$. However, according to the fact that the vehicle platform is driving only inside a testing facility, the approximation of the NED coordinate frame can be safely assumed as a global coordinate system.

2.5.2 Vehicle coordinate frames

There are three coordinate frames on the vehicle which are used for purpose of the trajectory tracking. These coordinate frames are *gps*, *base_footprint*

and *front_axle_footprint*.
The *gps* coordinate frame is positioned at the center of the Navio processing unit. All position measurements are related to that coordinate frame.
The *base_footprint* coordinate frame has its origin approximately at the center of gravity of the vehicle with its z-coordinate equal to zero.
The *front_axle_footprint* has its origin positioned in the center of the front axle with its z-coordinate equal to zero, which means that this coordinate frame shows a projection of the center of the front axle to the ground.

2.5.3 Transformations between global and vehicle coordinate systems

There are defined two coordinate frames to describe vehicle position in the global coordinate system.
The first coordinate frame is called *map*. This coordinate frame is identical to NED coordinate frame described in the section above. The second coordinate frame is called *odom*. The position of the vehicle in this frame is measured by the GPS sensor.

2.6 Sensor Fusion

The GNSS receiver has only limited measurement frequency $f_{GPS} = 10\text{Hz}$. This measurement rate doesn't have to be sufficient for attitude tracking. This problem is solved by the implementation of vehicle attitude estimation. The estimation algorithm uses a fusion of several sensors to estimate the attitude.
The estimation algorithm consists of three separate branches of cascaded complementary filters, one for each estimated value. The estimated values are the north position in local tangent plane N_{est} in meters, the east position in local tangent plane E_{est} and vehicle body heading ψ_{est} in radians.

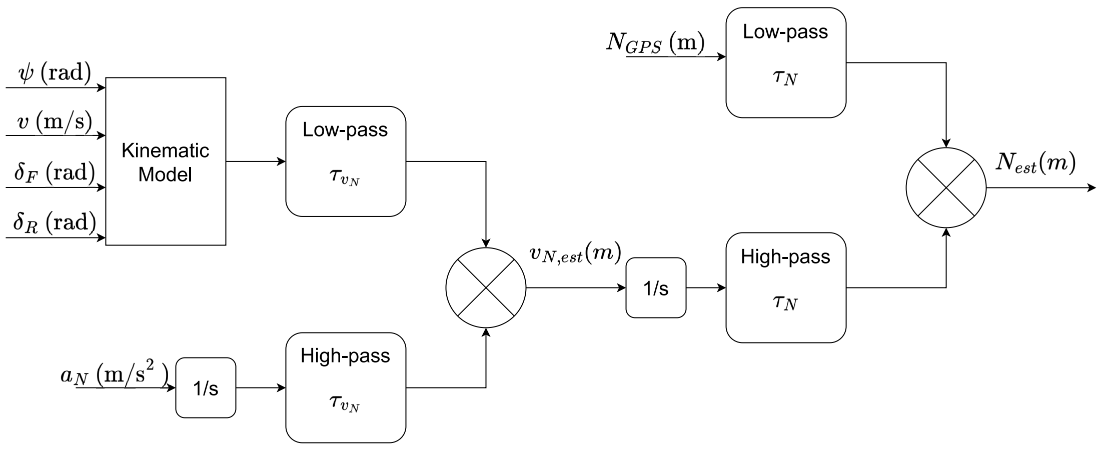


Figure 2.8: North coordinate estimation branch of the complementary filter

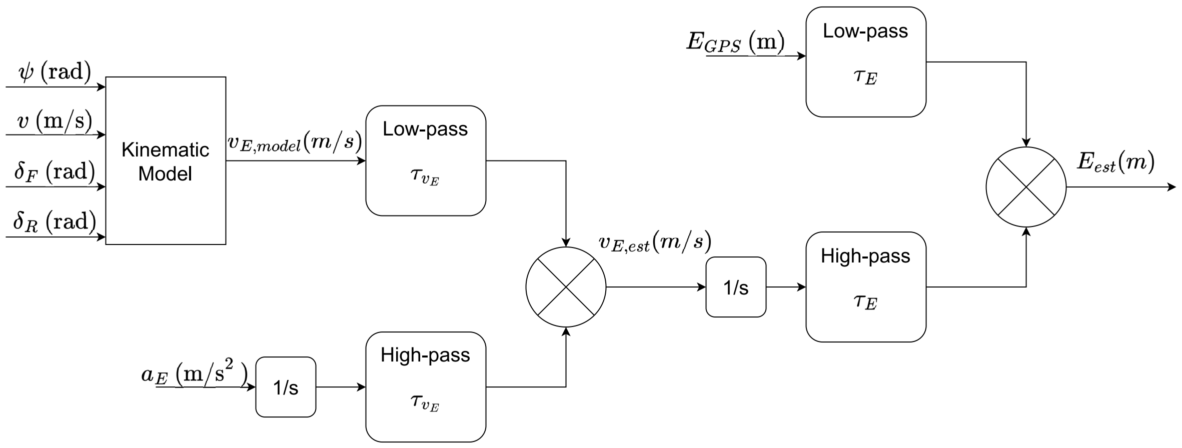


Figure 2.9: East coordinate estimation branch of complementary filter

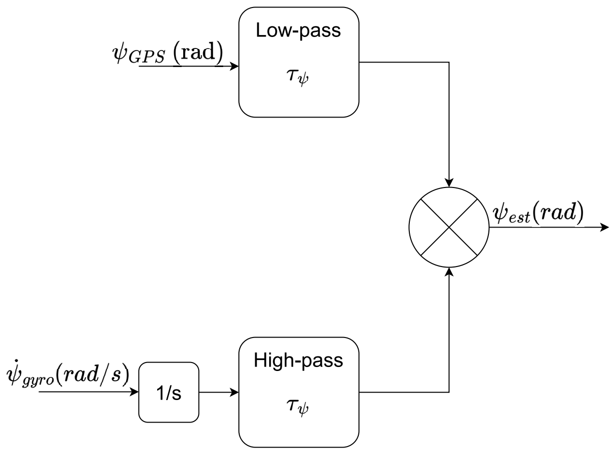


Figure 2.10: North coordinate estimation branch of the complementary filter

The branch for the vehicle heading, shown in figure 2.10, ψ_{est} estimation uses a gyroscope measuring rotation around the z-axis of the vehicle body $\dot{\psi}_{gyro}$ and absolute measurement of the vehicle body heading from the GNSS receiver ψ_{GPS}. This branch consists of only one complementary filter which combines the two signals using a low-pass filter and a high-pass filter. The heading complementary filter equations can be seen in the following equation

$$\begin{aligned}\Psi_{pgyro}(s) &= \frac{\tau_\psi s}{\tau_\psi s+1}\frac{1}{s}s\Psi_{gyro}(s), && (2.6)\\ \Psi_{pGPS}(s) &= \frac{1}{\tau_\psi s+1}\Psi_{GPS}(s), && (2.7)\\ \Psi_{filter} &= \Psi_{pgyro}(s)+\Psi_{pGPS}(s), && (2.8)\end{aligned}$$

where capital letters denotes laplacian images corresponding variables, $\Psi_{pgyro}(s)$ and $\Psi_{pGPS}(s)$ are outputs of corresponding filters and τ_{GPS} are tuning constants.
The branch for the north position N_{est} estimation uses accelerometers from the IMU unit, an absolute position given by the GNSS receiver, and velocity in the reference point at the vehicle calculated using the kinematic model from 3.2.1 from revolutions of front wheels. Data from the accelerometers are related to the vehicle body coordinate frame and have to be converted into a NED coordinate frame using the equation

$$\begin{aligned}a_N &= a_x\cdot\cos(\psi+\beta)-a_y\cdot\sin(\psi+\beta), && (2.9)\\ a_E &= a_x\cdot\sin(\psi+\beta)+a_y\cdot\cos(\psi+\beta), && (2.10)\end{aligned}$$

where a_N and a_E are accelerations corresponding to NED coordinate system axes, a_x and a_y are accelerations related to the vehicle body coordinate frame, ψ is the heading of the vehicle body and β is slip angle of the reference point at the vehicle. A similar transformation must be done for the velocity of the vehicle at reference point v_c. This transformation can be described using equations

$$\begin{aligned}v_N &= v_c\cdot\cos(\psi+\beta), && (2.11)\\ v_E &= v_c\cdot\sin(\psi+\beta), && (2.12)\end{aligned}$$

where v_N and v_E are velocities whose directions correspond to NED coordinate system axis, v_c is a value of the velocity at the reference point on the vehicle, ψ is the vehicle body heading and β is the side slip angle of the reference point on the vehicle calculated using the kinematic model.
This branch has two complementary filters connected in cascade, as can be seen in figure 2.8. The first filter is used to estimate the value of the velocity of the vehicle in the north direction. This filter uses the value of the acceleration of the vehicle in the north direction a_N and the value of velocity in the north direction v_N to return the estimated value of velocity in the north direction

$v_{N,est}$. The equation of the filter is then

$$\begin{aligned}
V_{paN}(s) &= \frac{\tau_{v_N} s}{\tau_{v_N} + 1} \frac{1}{s} A_N(s), && (2.13)\\
V_{pvN}(s) &= \frac{1}{\tau_{v_N} s + 1} V_{N,model}(s), && (2.14)\\
V_{N,est}(s) &= V_{pvN}(s) + V_{paN}(s), && (2.15)
\end{aligned}$$

where capital letters denote Laplace images, V_{pvN}, and V_{paN} are outputs of the corresponding filter, $V_{N,est}$ is the estimated value of the velocity of the vehicle in the north direction and τ_{v_N} is tuning constant.
The second filter in cascade uses the north position of the vehicle N and the estimated value of the velocity of the vehicle in the north direction $v_{N,est}$ and returns the estimated north position of the vehicle N_{est}. Filter mathematical description can be seen in the following equations.

$$\begin{aligned}
N_{pvN}(s) &= \frac{\tau_N s}{\tau_N + 1} \frac{1}{s} V_{N,est}(s), && (2.16)\\
N_{ppN}(s) &= \frac{1}{\tau_N s + 1} N(s), && (2.17)\\
N_{est}(s) &= N_{ppN}(s) + N_{pvN}(s), && (2.18)
\end{aligned}$$

where capital letters denote laplace images, N_{pvn}, and N_{ppN} are outputs of corresponding filters, N_{est} is estimated north position of the vehicle, and τ_N is a tuning constant.

Heading branch	Heading filter	τ_ψ	0.2
North branch	Position filter	τ_N	0.1
	Velocity filter	τ_{v_N}	0.75
East branch	Position filter	τ_E	0.1
	Velocity filter	τ_{v_E}	0.75

Table 2.1: Parameters of complementary filter

Chapter 3

Vehicle Model

3.1 Introduction

There were several mathematical vehicle models used for different purposes. All models used in this work are single-track models. This kind of vehicle model should be sufficient to properly describe the use vehicle platform because Ackermann steering geometry implemented in the platform allows for collapsing wheels on the same axle to the single wheel in the center of the axle without significant loss of precision of the model.

3.2 Modelling

3.2.1 Kinematic singletrack model

The kinematic single-track model is used as an odometry model for the complementary filter described in section 2.6. Inputs to the kinematic model are the steering angles on the front axle δ_F and on the rear axle δ_R in radians and the value of the velocity of the front wheels v_f. Outputs of this model are velocity in reference point on the vehicle v_c in meters per second, slip angle of reference point $beta$ in radians, yaw rate of vehicle $\dot{\psi}$ in radians per second, and curvature κ of a turning vehicle is doing. The outputs of the kinematic model could be described by equations

$$\begin{aligned}
\beta &= \arctan\left(\frac{l_r}{(l_f + l_r)}(\tan(\delta_f) - \tan(\delta_r)) + \tan(\delta_r)\right), && (3.1)\\
\dot{\psi} &= v_f \cdot frac \sin(\delta_f - \delta_r)(l_f + l_r)\cos(\delta_r), && (3.2)\\
v_c &= v_f \cdot \frac{\cos(\delta_f)}{\cos\beta}, && (3.3)\\
\kappa &= \frac{\sin(\delta_f - \beta)}{l_f \cdot \cos(\delta_f)} && (3.4)
\end{aligned}$$

3.2.2 Nonlinear single-track model

The nonlinear single-track model used in this work was adopted from [EKHH20]. This model extends the functionality of the kinematic single-track model by taking into account the dynamics of tires or dissipative forces affecting a vehicle. Due to the complexity of the nonlinear model of the vehicle dynamics, the equations won't be shown here. The linear lateral model described in the next section is derived from this nonlinear model.

3.2.3 Linear Lateral Dynamics Model

The nonlinear single-track vehicle model described in 3.2.2 can be used to derive the linear model, which assumes only the lateral dynamics of the vehicle. Derivation of the linear model is described in [EZKH21].
The linear approximation of lateral dynamics is modeling only side slip angle β in the center of gravity of the vehicle and yaw rate of the vehicle body $\dot{\psi}$. This model can be described by the equation

$$\begin{pmatrix}\dot{\beta}\\ \ddot{\psi}\end{pmatrix} = \begin{pmatrix} -\frac{C_f+C_r}{mv} & \frac{l_rC_r-l_fC_f}{mv^2}-1 \\ \frac{l_rC_r-l_fC_f}{I_z} & -\frac{l_f^2C_f+l_r^2C_r}{vI_z}\end{pmatrix}\begin{pmatrix}\beta\\ \dot{\psi}\end{pmatrix} + \begin{pmatrix}\frac{C_f}{mv} & \frac{C_r}{mv}\\ \frac{l_fC_f}{I_z} & -\frac{l_rC_r}{I_z}\end{pmatrix}\begin{pmatrix}\delta_f\\ \delta_r\end{pmatrix}. \quad (3.5)$$

This model is a linear parameter-varying model. The parameter varying in this model is velocity v. Parameters l_f and l_r are distances of axles from the center of gravity, I_z is the moment of inertia of the vehicle body with respect to the z-axis and m is the mass of the vehicle. The parameters C_f and C_r are cornering stiffnesses of the front and rear tires. These parameters are slopes of the bilinear approximation of the Pacejka simplified lateral tire model and can be obtained using the equation

$$\begin{aligned} C_r &= D_{yr} \cdot B_{yr} \cdot C_{yr} \cdot F_{zr}, & (3.6)\\ C_f &= D_{yf} \cdot B_{yf} \cdot C_{yf} \cdot F_{zf}, & (3.7) \end{aligned}$$

where D, B, and C are respective tire parameters and F_{zr}, and F_{zf} are forces affecting tires in z-axis direction.
This model in its discrete form is further used as a predictor for attitude control using MPC algorithm.

3.3 Identification

The identification of vehicle parameters was done only for the linear lateral vehicle model. The parameters m, l_f, and l_f were identified by direct measurement of the vehicle platform. The parameters I_z, C_f and C_r were identified by fitting system response to measured data. This identification of parameters was done by minimizing the deviation of states trajectory from measured data. The optimization problem can be written as

$$\min_{p\in\mathbb{R}^3} J(p) = \min_{p\in\mathbb{R}^3}(y(t) - \int_{t_0}^{t_1} f(t,x,u,p)dt)^2 \tag{3.8}$$

where $p = \begin{pmatrix} I_z & C_f & C_r \end{pmatrix}^T$ is parameter vector, $y(t)$ is representing measured data and $f(t,x,p)$ is function of parameters

$$f(t,x,u,p) = \mathbf{A}x + \mathbf{B}u \tag{3.9}$$

where $\mathbf{A}$ is system matrix from equation 3.5, $\mathbf{B}$ is input matrix from equation 3.5, $x = \begin{pmatrix} \beta & \dot{\psi} \end{pmatrix}^T$ is vector of states and $u = \begin{pmatrix} \delta_r & \delta_f \end{pmatrix}^T$ is vector of inputs.
Data for identification were obtained by performing a so-called crab walk. This maneuver is done with both steering axles turning about the same angle. This maneuver then ends up with the side slip angle being the same as a steering angle on axles and only very small or short time deviations of the yaw rate of the vehicle. Experimental data can be seen in figure 3.1 and inputs used to generate these data can be seen in figure 3.2.

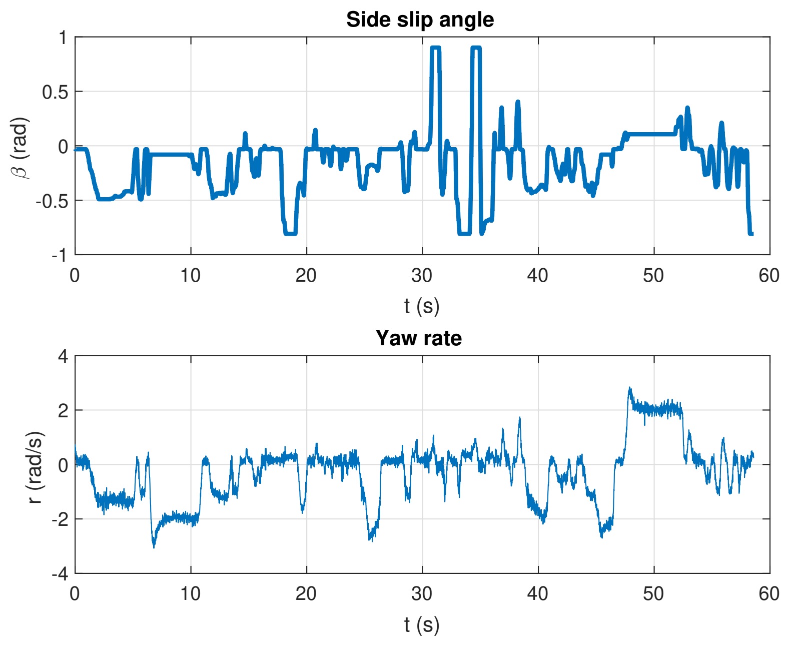


Figure 3.1: Lateral identification data

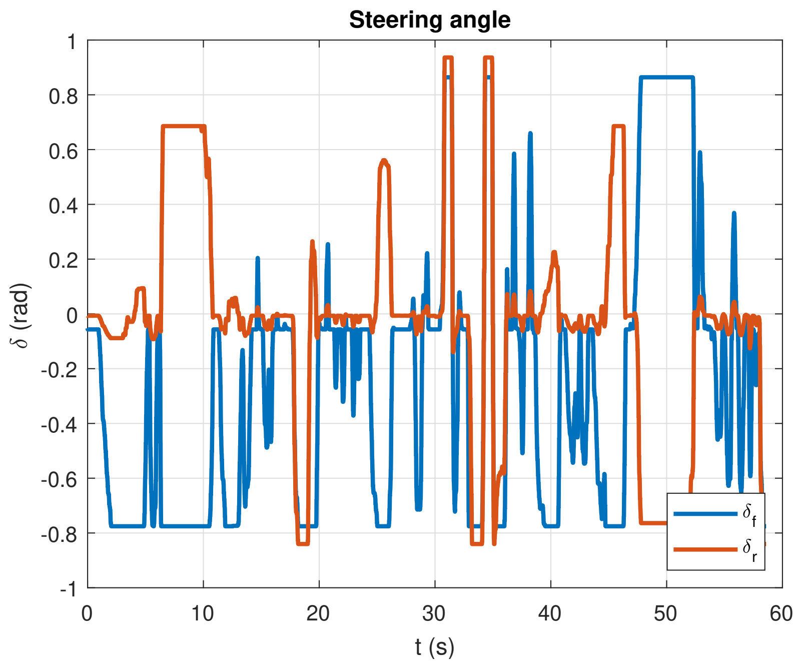


Figure 3.2: Inputs for lateral identification

Identified parameters can be seen in table 3.1. Figure 3.3 shows a comparison of the response of the identified linear system and measured data.

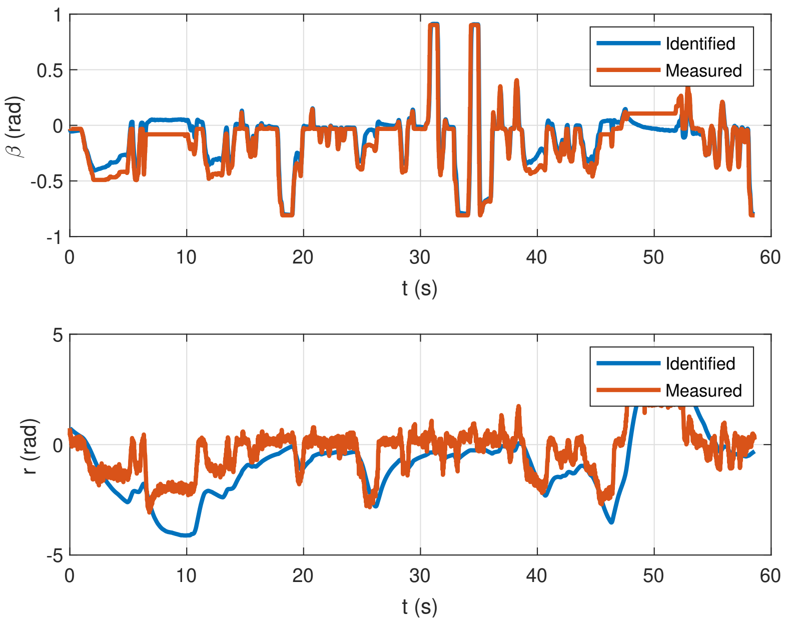


Figure 3.3: Comparison of identified system response and measured data

l_f [m]	0.3	l_r [m]	0.3
m [kg]	21	I_z [kg·m^2]	85.334
C_f [-]	1000	C_r [-]	1000

Table 3.1: Identified parameters of linear parameter varying model

3.3.1 Modified Experiment

The data used from the first identification attempt didn't include slips of the vehicle and also the experiment was done with a lower velocity of the vehicle. Under these circumstances, the model couldn't be properly identified. The second experiment was performed at a higher speed and the steering was more aggressive than in the experiment. The data from the second experiment included slips of the tires and not only a slip of the vehicle body. The parameters identified from the data from the second experiment can be seen in table 3.2.

l_f [m]	0.3	l_r [m]	0.3
m [kg]	21	I_z [kg·m^2]	1.2562
C_f [-]	53.3964	C_r [-]	68.8640

Table 3.2: Modified parameters of linear parameter varying model

Unfortunately, these new data couldn't be used as data for the MPC predictor because there was no time left to perform new experiments.

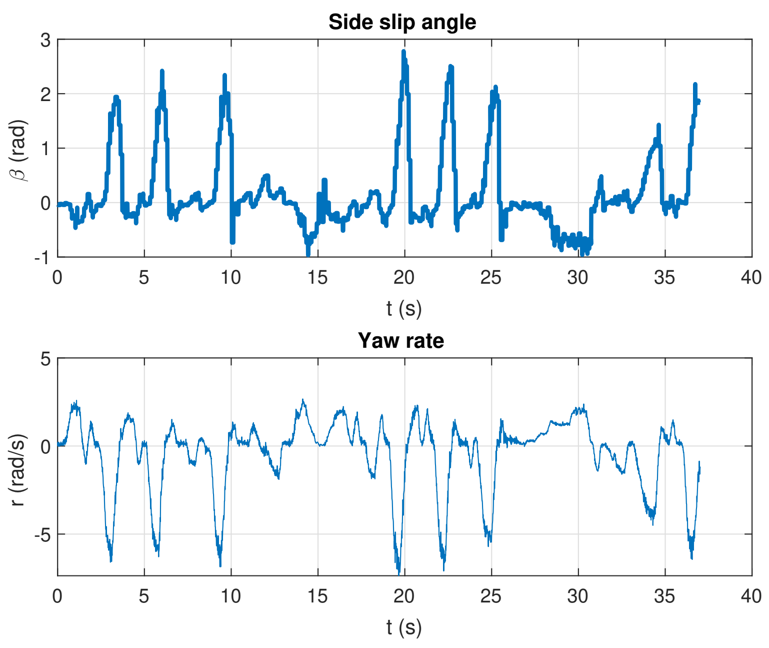


Figure 3.4: Lateral identification data

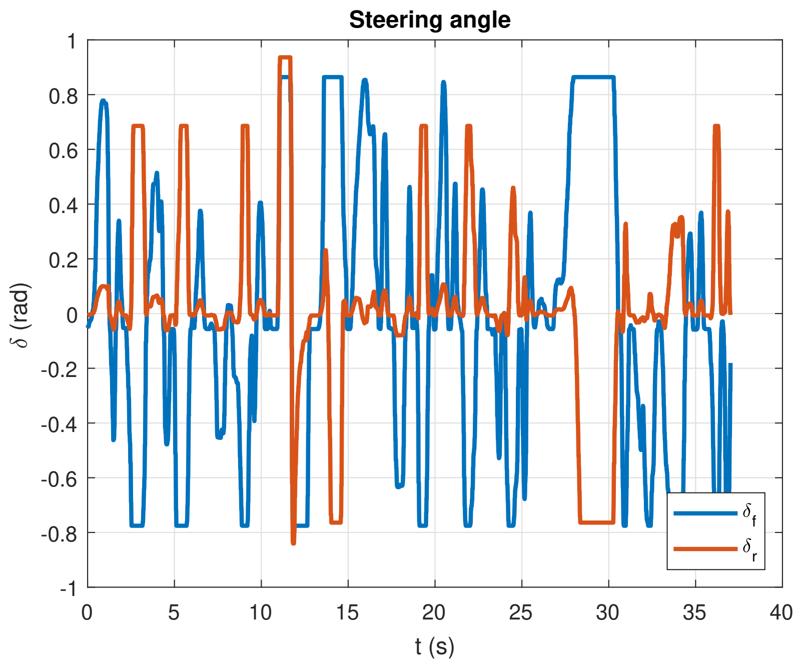


Figure 3.5: Inputs for lateral identification

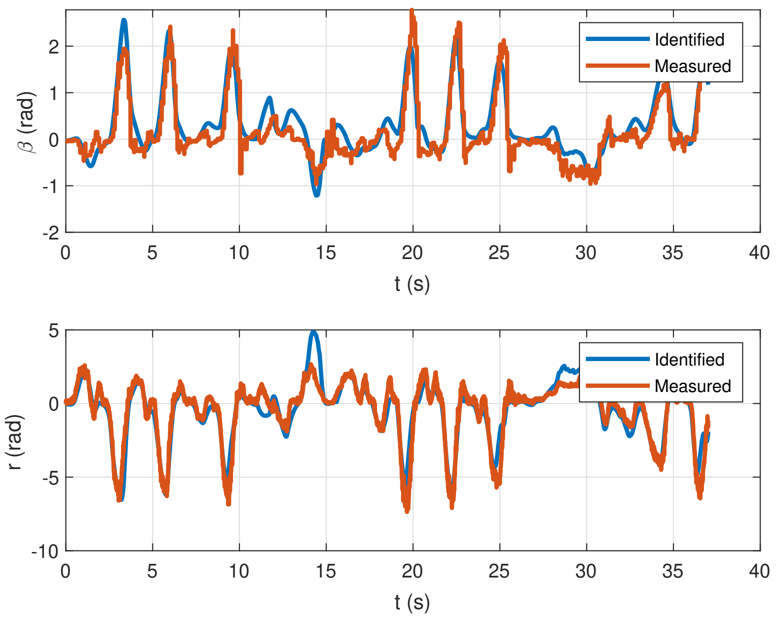


Figure 3.6: Comparison of identified system response and measured data

3.4 Vehicle Dynamics Simulator

To simulate the algorithms there were used a HIL simulator built in ROS2 environment. The simulator was implemented as a ROS2 node and was run on the Jetson Xavier. This simulator had the interface built according to the real ROS2 node which operates the vehicle to be as close to it as possible. This simulator was used to test the interface of the algorithms before they were deployed on the vehicle platform and also to check the behavior of the algorithm. The visualization tool used with the simulator was a ROS2 built-in package named RViz (`https://github.com/ros2/rviz`), which has the capability of visualizing the ROS2 messages in real time.
The mathematical model used as a simulation model was composed of the LPV model described in section 3.2.3, the nonlinear transformation of the polar velocity to x-axis and y-axis velocities and first-order velocity system approximation. The equation describing the simulation model can be seen in

the equations below.

$$\begin{pmatrix} \dot{\beta} \\ \ddot{\psi} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} -\frac{C_f+C_r}{mv} & \frac{l_rC_r-l_fC_f}{mv^2} - 1 & 0 \\ \frac{l_rC_r-l_fC_f}{I_z} & -\frac{l_f^2C_f+l_r^2C_r}{vI_z} & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \beta \\ \dot{\psi} \\ \psi \end{pmatrix} \quad (3.10)$$

$$+ \begin{pmatrix} \frac{C_f}{mv} & \frac{C_r}{mv} \\ \frac{l_fC_f}{I_z} & -\frac{l_rC_r}{I_z} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_f \\ \delta_r \end{pmatrix} \quad (3.11)$$

$$\dot{x} = \cos(\beta + \psi) \cdot v \quad (3.12)$$

$$\dot{y} = \sin(\beta + \psi) \cdot v \quad (3.13)$$

$$\dot{v} = a \cdot v + b \cdot u \quad (3.14)$$

Chapter 4

Path Tracking Architecture

4.1 Overall Architecture

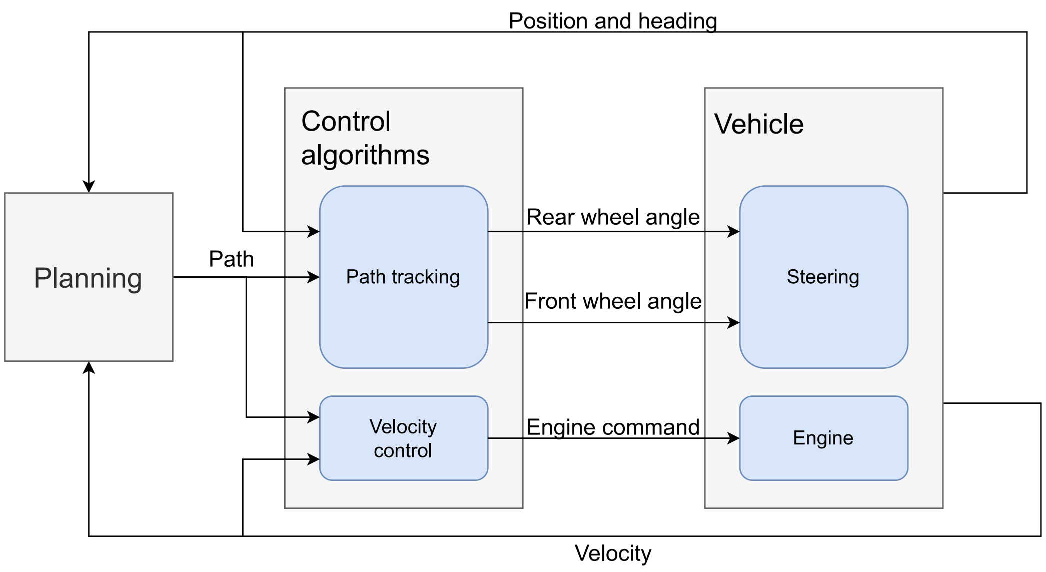


Figure 4.1: Control layout block diagram

Vehicle control was divided into two separate control tasks as can be seen in figure 4.1.
The first part is the velocity controller, which takes the reference velocity from the planning algorithm and outputs the engine command in a form of

PWM width.
The second part is the path tracking algorithm, which takes the $[N, E]$ position and heading as a reference from the planning algorithm and outputs a control signal in a form of steering angles of the front and rear axle.

4.2 Planning and Control Interface

4.2.1 Path Planning

Path planned by planning algorithm is an ordered set of points $p_k = \begin{pmatrix} N_k & E_k & \psi_k & v_k \end{pmatrix}^T$, these values are then used as reference signals for the attitude control algorithm and velocity control algorithm. N_k and E_k are positions in the NED coordinate system of the local tangent plane namely N_k is a north position in the local tangent plane in meters and E_k is an east position in the local tangent plane in meters. Both position coordinates are relative to the origin of the local tangent plane approximation described in section 2.5.1. Heading ψ_k gives desired heading of the vehicle to get from point p_k to p_{k+1} taking a straight line. The size of the velocity v_k is the velocity for which the path from point p_k to point p_{k+1} was planned.

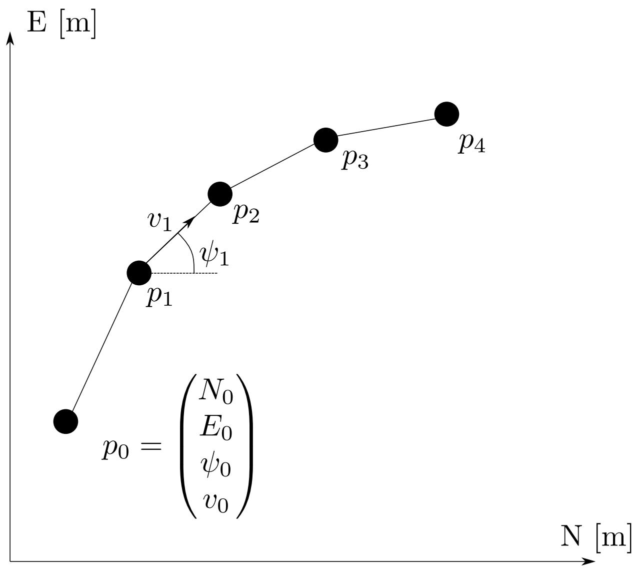

E [m]
N [m]
p_4
p_3
p_2
v_1
ψ_1
p_1
$p_0 = \begin{pmatrix} N_0 \\ E_0 \\ \psi_0 \\ v_0 \end{pmatrix}$

Figure 4.2: Reference path visualization

4.2.2 Integration

Planning and control algorithms are implemented as separate nodes inside the ROS2 network. The path planning node is deployed on NVIDIA Jetson AGX Xavier and the path tracking algorithm and velocity controller are deployed on Raspberry Pi 4B. These two computational units are connected via Ethernet and communication between them is established by the ROS2 layer. Path planning algorithm sends Path filled with information mentioned above in section 4.2.1. Path message [PAT] is an inbuilt message of the ROS2 environment, namely *nav_msgs* package. The definition of the path message can be seen below.

Message name	Element	Type
Path		
	header	Header
	poses	Array of PoseStamped

Table 4.1: Path message definition

4.3 Vehicle and Control Interface

4.3.1 Position Control

The path-tracking algorithms control the steering angle of both the rear and the front axle. The output of the path tracking algorithm is a request on the steering angle on the given axle in radians, δ_F for the front axle and δ_R for the rear axle. Maximal allowed steering angle deviation of both axles is $|\delta_{max}| = \frac{\pi}{6}$ rad. These steering angles sent by the attitude control algorithm are then received by the node controlling a vehicle, converted to PWM width value, and sent to servo motors controlling the steering angles of particular axles.
These steering angle requests are sent in ROS2 message called *Pose2D* (ROS message definition `http://docs.ros.org/en/noetic/api/geometry_msgs/html/msg/Pose2D.html`). The definition of this message could be seen in the table below.

Message Name	Elements	Types
Pose2D		
	x	float64
	y	float64
	theta	float64

Table 4.2: Definition of Pose2D message

In this particular case *Pose2D* message's element x was used to transfer front axle steering angle δ_F and element y was used to transfer rear axle steering angle δ_R. Element *theta* was left unassigned and not used.

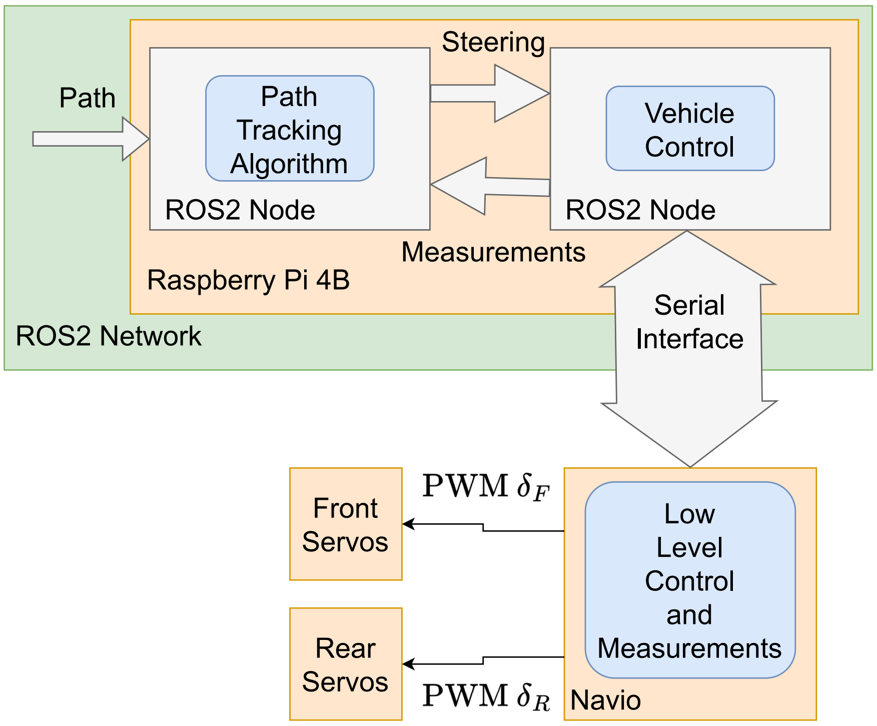


Figure 4.3: Block diagram of the interface between position control and vehicle hardware

4.3.2 Velocity Control

The control action u_D outputted from the velocity controller is the command that is directly proportional to the width of the PWM signal applied to the motor controller. This signal is in the range $u_D \in [-100, 100]$. The control signal from the velocity controller is sent wrapped in a ROS2 message to the ROS2 node communicating with the hardware of the vehicle. The message used for this is the *Twist* (`http://docs.ros.org/en/melodic/api/geometry_msgs/html/msg/Twist.html`).

Message name	Element	Type
Twist		
	linear	Vector3
	angular	Vector3

Table 4.3: Definition of *Twist* message

Message name	Element	Type
Vector3		
	x	float64
	y	float64
	z	float64

Table 4.4: Definition of *Vector3* message

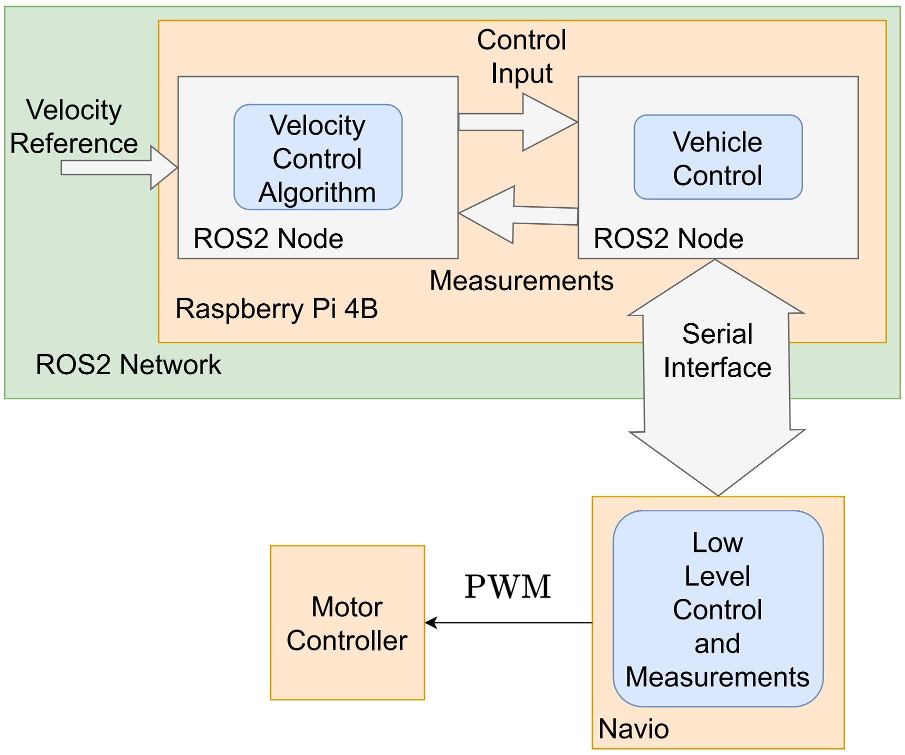


Figure 4.4: Block diagram of the interface between velocity control and vehicle hardware

4.3.3 Integration

Communication between control algorithms and the vehicle is also established by the ROS2 layer. As mentioned in chapter 2.3.1 manipulation of vehicle actuators is also implemented as a ROS2 node. These two nodes are deployed on the same computational unit. ROS2 nodes in which the velocity controller and path tracking algorithm are implemented communicate with ROS2 node that manipulates vehicle actuators namely the engine and steering mechanism.

Chapter 5

Velocity Controller

5.1 Approximation of longitudinal dynamics

Observation of the data shown in figure 5.1 shows that transfer from input u_D to velocity v can be safely approximated using a first-order linear system.

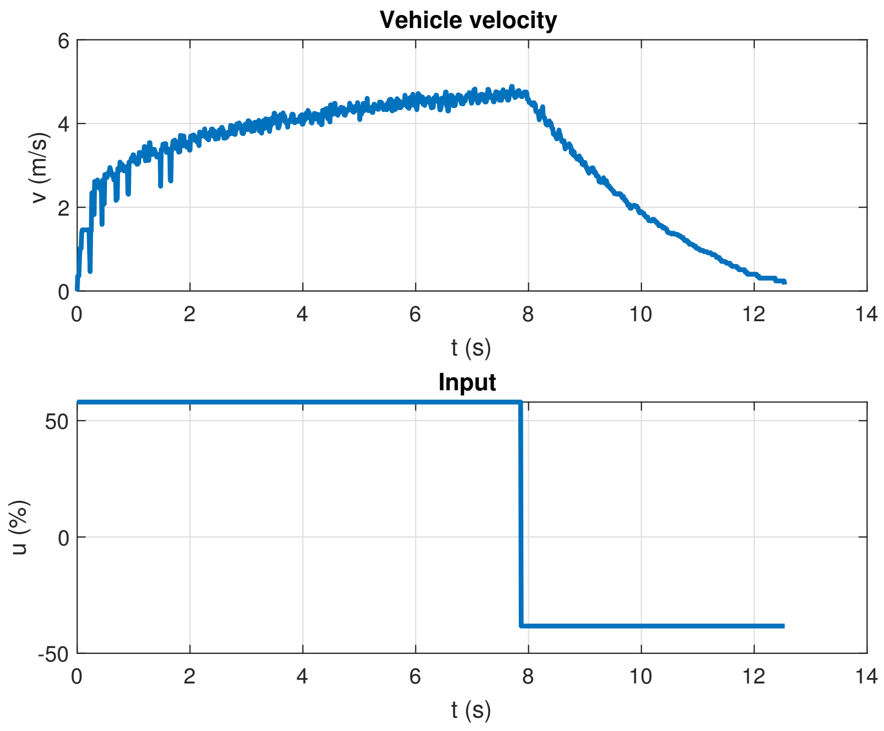


Figure 5.1: Measured velocity and engine input

Longitudinal dynamics was only considered as a transfer from input u_D mentioned in section 4.3 to vehicle velocity in a reference point. We can assume the transfer function of longitudinal dynamics approximation to proper transfer without zeros. The equation could be seen 5.1.

$$H_{long}(s) = \frac{k}{s+a} \tag{5.1}$$

The transfer was then converted to a state space description

$$\dot{v}(t,x) = -a \cdot v(t,x) + k \cdot u_D(t) = f(t,x,c), \tag{5.2}$$

where $c = \begin{pmatrix} a & k \end{pmatrix}^T$ is a parameter vector and t is time in seconds. This system was identified using the least-squares fitting method. That means that model was fitted to the data acquired by doing experiments on the vehicle. Parameters were then obtained by minimization of the equation

$$\min_{c\in\mathbb{R}^2} J(c) = \min_{c\in\mathbb{R}^2}(y(t) - \int_{t_0}^{t_1} f(t,x,c)dt)^2. \tag{5.3}$$

This minimization problem is formulated as a nonlinear least squares problem and was solved with the use of MATLAB function called *lsqnonlin* [Mata]. Data used for identification were generated using engine input $u_D = 58\%$ and zero steering angle. Experimental data can be seen in figure 5.1.

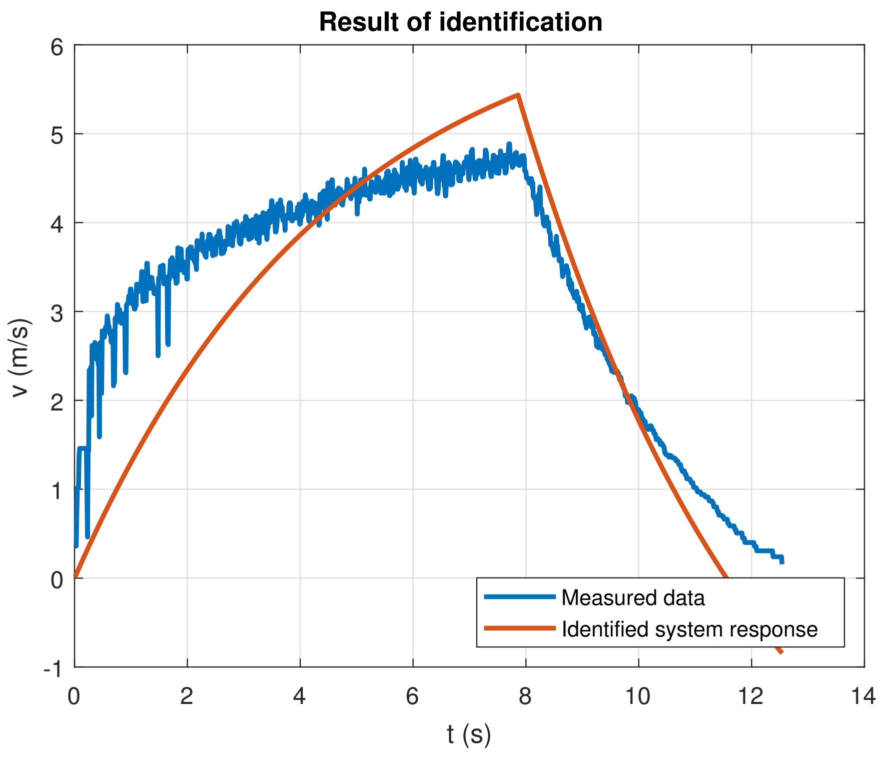


Figure 5.2: Result of first-order approximation identification

Identified parameters are

$$a = 0.9144, \tag{5.4}$$
$$k = 0.0690. \tag{5.5}$$

This first-order approximation was then directly used to design a velocity controller.

5.2 Controller design

The design of the controller was done using the root locus method with a forced controller structure. It was decided to use a PI controller because this type of controller is capable of holding steady-state error at zero and also

provides some degree of robustness. This could help with inaccuracies in a model used for the design of the controller. The parameters tuned on the design model can be seen in table 5.1.
The resulting regulator was then tuned using tests on the real platform to meet the requirements.
However, these parameters were too aggressive for the real system. The

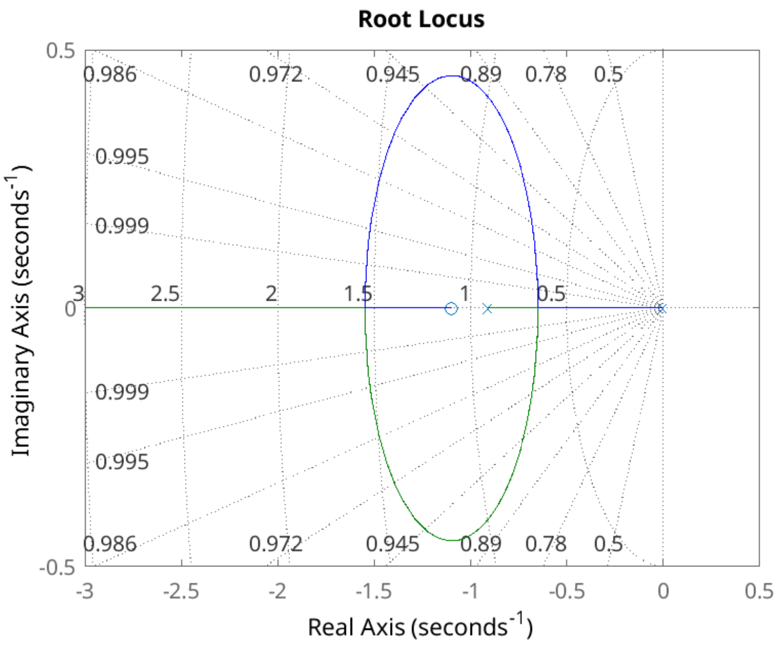


Figure 5.3: Root locus plot for the approximation of system longitudinalal dynamics with forced PI regulator structure

parameters were tuned using experiments with the real system. These parameters can be seen in table 5.2

K_p	31.65
K_i	34.78

Table 5.1: Parameters of the PI controller tuned on design model

K_p	23
K_i	15

Table 5.2: Parameters of the PI controller tuned for design model

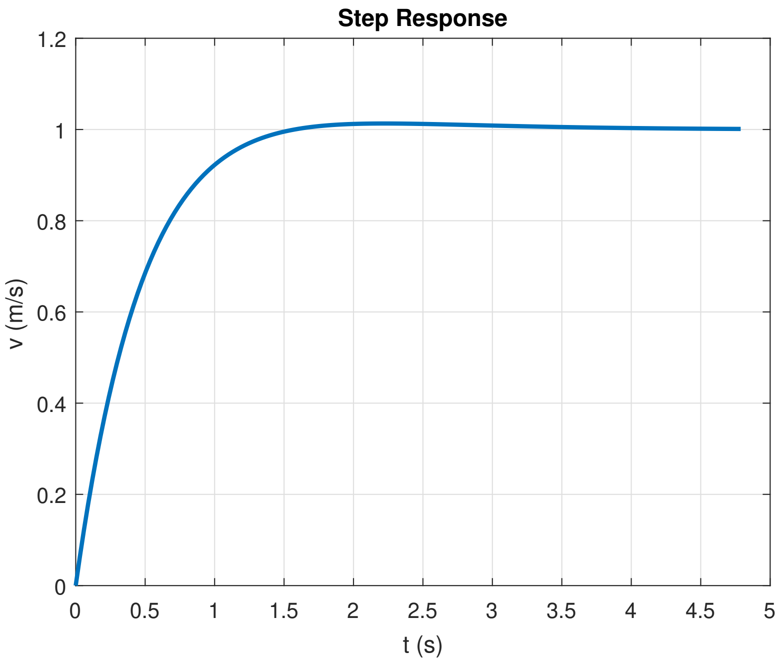


Figure 5.4: Step response of controlled design model

5.3 Velocity reference

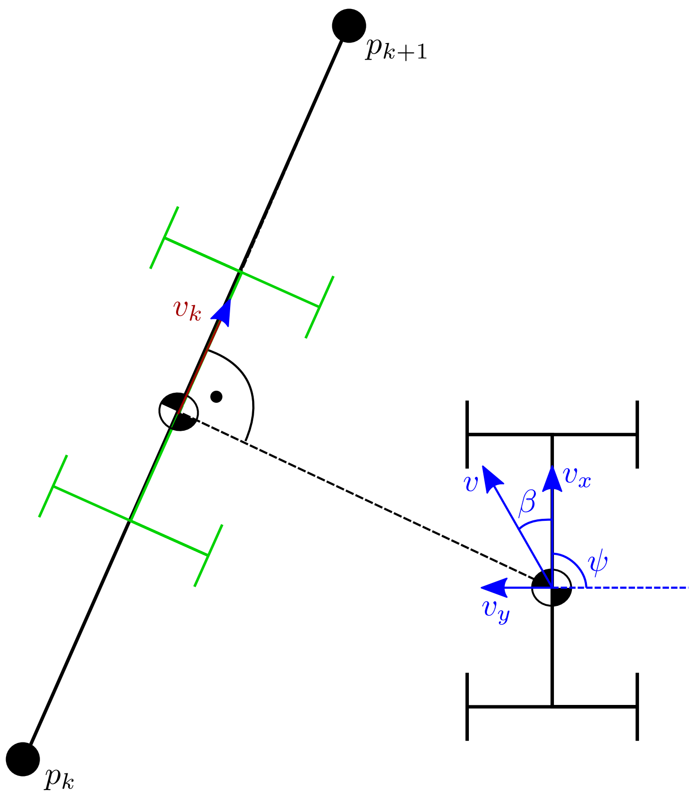


Figure 5.5: Shadow vehicle projection

The problem with velocity while tracking trajectory is that velocity given by time gaps between points on the trajectory is assumed in case the vehicle is moving precisely along the planned trajectory. This means that if the vehicle deviates from the given path the velocity reference has to be modified.
This can be solved by introducing a so-called "shadow" vehicle, which is a projection of the vehicle onto the trajectory, as can be seen in figure 5.5. The shadow vehicle projection can be used to calculate the velocity of the real vehicle necessary to achieve point p_{k+1} in time. When a vector of velocity is projected on the direction vector line segment given the position of points p_k and p_{k+1}, it gives the size of the velocity vector v_g of the shadow vehicle moving along the trajectory. The size of velocity vector v_g then becomes a controlled value and planned velocity v_k between points p_k and p_{k+1} is taken as a reference value.

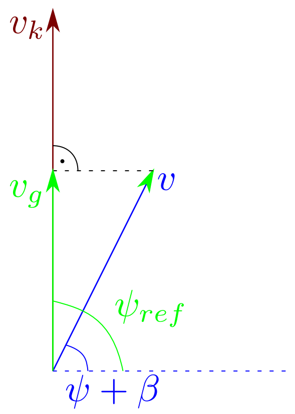


Figure 5.6: Velocity projection

The projection can be calculated using the following formula

$$|v_g| = \frac{v_k^T \cdot v}{|v_k|}, \tag{5.6}$$

which gives us the size of the velocity vector of the shadow vehicle v_g.

5.4 Rate Limiter

The velocity controller was supplemented with a rate-limiting feature. This feature is changing the step change of the reference to the linear change with a given slope.
The rate limiter is designed to limit the acceleration of the vehicle to $a_{max} = 1\,\mathrm{ms}^{-2}$. The velocity control algorithm runs with sampling period $T_{cc} = 0.01$ s. The rate-limiting feature was implemented as a software feature described by the algorithm 1.

Algorithm 1 Rate limiter algorithm

```
if New reference available then
    $v_{rate} \leftarrow$ Measured velocity
    $v_{ref} \leftarrow$ New reference velocity
end if
if $v_{ref} - v_{rate} \leq 0$ then
    $v_{rate} \leftarrow v_{ref}$
end if
while $|v_{rate} - v_{ref}| <= \epsilon$ do
    $v_{rate} \leftarrow v_{rate} + \gamma$
end while
```

where v_{rate} is the velocity reference outputted by the rate limiter algorithm, v_{ref} is the external velocity reference and γ is the constant increase. In our case $\gamma = 0.01\,\mathrm{ms}^{-1}$ due to the while loop having the sample period $T_{cc} = 0.01$ s and frequency $f_{cc} = 100$ Hz the rate limiting algorithm will limit the rate of change of the reference velocity to $a_{max} = 1\,\mathrm{ms}^{-2}$.

Chapter 6

Path Tracking

Algorithms designed to track trajectory are mainly controllers with some level of predictivity. In trajectory tracking problems predictivity of controllers could be crucial when performing certain maneuvers, for example driving through a turn with a high value of curvature at high speed. Reactive controllers like basic PID can have a slower reaction and cause the vehicle to deviate too much from the path and it might crash into the crash barrier alongside the road. A good analogy to using reactive controllers for a path tracking problem could be driving a vehicle and looking only into a side mirror, the driver will know that there is a turn just when he actually passes the turn.

6.1 Stanley Control Law Inspired Algorithm

This controller is highly inspired by Stanley Control Law used in the publication [HTMT07].
Stanley control law is consisting of several parts namely heading part ψ_{ctrl}, cross-track part d_{ctrl}, and curvature part r_{ctrl}, this can be understood as a yaw damper, and steering part s_{ctrl}. The full equation of Stanley control law

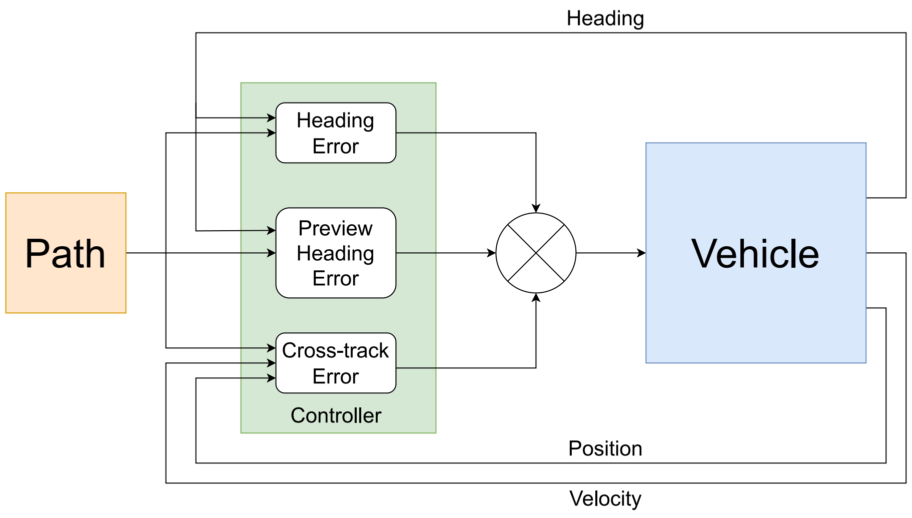


Figure 6.1: Diagram of Stanley Control Law inspired algorithm

used can be written as

$$\delta(t) = \underbrace{(\psi_{ref} - \psi)}_{\psi_{ctrl}} + \underbrace{\arctan \frac{ke(t)}{k_{soft} + v(t)}}_{d_{ctrl}} + \\ + \underbrace{k_{d,yaw}(r_{meas} - r_{traj})}_{r_{ctrl}} + \underbrace{k_{d,steer}(\delta_{meas}(i) - \delta_{meas}(i+1))}_{s_{ctrl}}, \tag{6.1}$$

where $\delta(t)$ is the steering angle calculated with control law, ψ_{ref} is the reference heading, ψ is the measured vehicle heading, $e(t)$ is a cross-track error, k_{soft} is softening constant, which prevents division by zero, r_{meas} is yaw rate of the vehicle, r_{traj} is estimated yaw rate of reference trajectory, δ_{meas} is discrete time measurement of steering, i is time index of measurement of steering one period earlier and k, $k_{d,yaw}$ and $k_{d,steer}$ are tuning constants.
The architecture of the baseline controller used in this work is highly inspired by Stanley control law. When designing this controller, curvature part r_{ctrl} and steering part s_{ctrl} were removed, because the vehicle platform doesn't allow measurement of the steering angle of axles. The designed baseline algorithm was supplemented with the predictive part in form of lookahead measurements of the vehicle heading.

The equation of this controller could be written as

$$\delta(t) = \underbrace{k_{yaw}(\psi_{ref} - \psi)}_{\psi_{ctrl}} + \underbrace{\arctan \frac{ke(t)}{k_{soft} + v(t)}}_{d_{ctrl}} + \underbrace{k_{lh}(\psi_{lh} - \psi)}_{\psi_{lh,ctrl}}, \tag{6.2}$$

where ψ is measured heading of vehicle, ψ_{ref} is heading reference in current position, ψ_{lh} is heading reference in lookahead distance, $e(t)$ is cross-track error, k_{soft} is softening constant, $v(t)$ is vehicle velocity in reference point on vehicle and k and k_{lh} are tuning constants. Additionally, Stanley Control Law output action is restricted only to the allowed range $\left[\delta_{min}, \delta_{max}\right]$ so the final output of the controller is described by equation 6.3.

$$\delta_F(t) = sat_{\delta_{min}}^{\delta_{max}}(\delta(t)) \tag{6.3}$$

where δ_F is steering angle on front axle, δ is control command from modified Stanley algorithm and $\text{sat}_{\delta_{min}}^{\delta_{max}}$ is saturation operator with upper bound in δ_{max} and lower bound in δ_{min}.

6.1.1 Predictive part

The predictive part in this controller is designed using lookahead distance dependent on the vehicle velocity. Calculation of the lookahead point is done using constant time gap t_{GAP} between vehicle and lookahead point and ghost vehicle velocity defined in section 5.3.

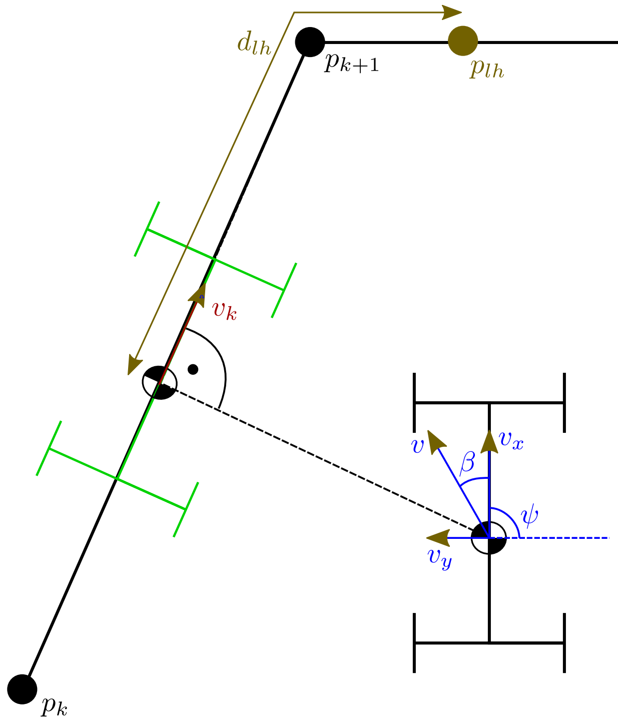


Figure 6.2: Lookahead point

Calculation of lookahead point distance is done using the equation

$$d_{lh} = d_0 + |v_g| \cdot t_{GAP}. \tag{6.4}$$

the lookahead point is then found in distance d_{lh} "walked" along the path, $|v_g|$ is the size of the velocity vector and d_0 is the offset constant. The reference heading of the lookahead part of the controller is a reference of the path segment given by points p_i and p_{i+1} to which the lookahead point p_{lh} belongs.

6.1.2 Error measurement

Calculation of heading error and lookahead heading error is done using the following equations

$$\psi_{err} = \psi_{ref} - \psi, \tag{6.5}$$
$$\psi_{lh,err} = \psi_{lh} - \psi, \tag{6.6}$$

where psi_{err} is current heading error and $\psi_{lh,err}$ is heading error in distance of lookahead point.
However, the calculation of cross-track error could be a bit tricky. Cross-track error when considering smooth trajectory is defined as the length of the line segment between the closest point on the path and the reference point on the vehicle, which is also perpendicular on the line tangential to the closest point. However, cross-track error calculation, when considering trajectory as a set of positions connected by line segments as in this case, can be done using vector rejection, which will calculate the size of vector $\vec{e}$ using vectors $\vec{p}$ and $\vec{t}$ as is shown in figure 6.3.

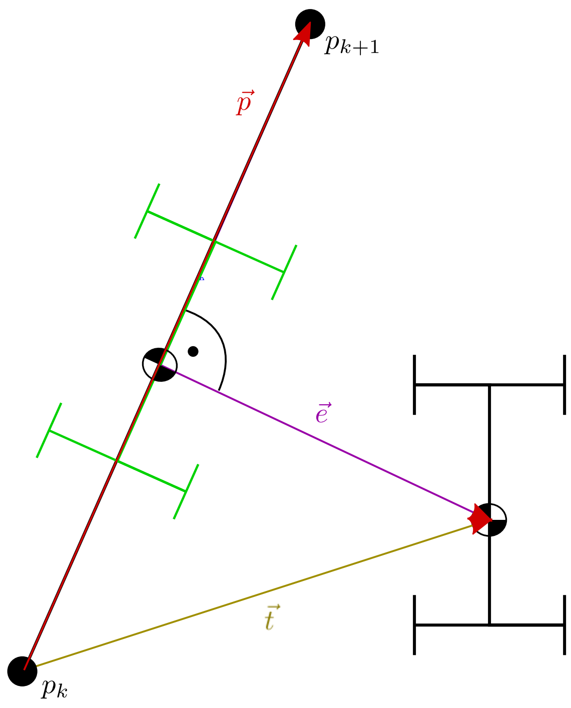


Figure 6.3: Calculation of cross-track error

Rejection of vector $\vec{t}$ from vector $\vec{p} = \begin{pmatrix} p_x & p_y \end{pmatrix}^T$ can be calculated as projection of vector $\vec{t}$ on vector $\vec{p}^{\perp} = \begin{pmatrix} p_y & -p_x \end{pmatrix}^T$ perpendicular to vector $\vec{p}$. The equation of this projection is then

$$|\vec{e}| = \frac{\vec{t} \cdot (\vec{p}^{\perp})^T}{|\vec{p}^{\perp}|} \tag{6.7}$$

This calculation gives the signed cross-track error so that the cross-track error is negative when the path is closer to the right side of the vehicle and positive if the path is closer to the left side of the vehicle.

6.2 Model Predictive Control

6.2.1 Basic tracking problem

The basic tracking problem is that it is given the desired heading and the desired position of the vehicle by waypoints on the reference path. The basic problem is to track these references directly. To be able to do this the predictor has to be composed of several parts.
The first part is the linear parameter variable model introduced in section 3.2.3.
The LPV model can be then extended with the integrator of the heading. This extended model can be expressed by equation 6.8.

$$\begin{pmatrix} \dot{\beta} \\ \ddot{\psi} \\ \dot{\psi} \end{pmatrix} = \begin{pmatrix} -\frac{C_f+C_r}{mv} & \frac{l_rC_r-l_fC_f}{mv^2} - 1 & 0 \\ \frac{l_rC_r-l_fC_f}{I_z} & -\frac{l_f^2C_f+l_r^2C_r}{vI_z} & 0 \\ 0 & 1 & 0 \end{pmatrix} \begin{pmatrix} \beta \\ \dot{\psi} \\ \psi \end{pmatrix} + \begin{pmatrix} \frac{C_f}{mv} & \frac{C_r}{mv} \\ \frac{l_fC_f}{I_z} & -\frac{l_rC_r}{I_z} \\ 0 & 0 \end{pmatrix} \begin{pmatrix} \delta_f \\ \delta_r \end{pmatrix}. \tag{6.8}$$

The body heading of the vehicle can be directly obtained by integration of yaw rate $\dot{\psi}$ and in addition, the equation describing the integration is also linear.
The velocity of the vehicle could be broken down to the north and the east part of the velocity using equation 6.9.

$$\dot{N} = \cos(\beta + \psi) \cdot v, \tag{6.9}$$
$$\dot{E} = \sin(\beta + \psi) \cdot v, \tag{6.10}$$

where $\dot{N}$ is the north part of the velocity, $\dot{E}$ is the east part of the velocity, v is the value of the velocity at the reference point on the vehicle, β is side slip angle of the reference point on the vehicle, and ψ is body heading of the vehicle. The equations 6.9 have to be linearized to use these equations

in the linear predictor. The lateral linear model is already the LPV model parametrized by the variable velocity of the vehicle v, thus it is possible to approach the linearization in the same way and also take the velocity as a variable parameter.
This system although has no equilibrium if holds that $v > 0$, so it has to be linearized along the state trajectory. The system linearized in this way will have a form described in equation 6.11.

$$\begin{aligned} \dot{x}_p + \Delta\dot{x} &= \mathbf{A}_p \Delta x + \mathbf{B}_p \Delta u, && (6.11)\\ \Delta x &= x - x_p && (6.12)\\ \Delta u &= u - u_p && (6.13) \end{aligned}$$

where $\dot{x}_p$ is a vector that describes the direction of the state development, $\Delta\dot{x}$ is a vector that describes the direction in which states deviate from the trajectory, Δx is a vector of state increases, x_p is the operational point and Δu is the input vector increase and because the input operational point u_p is equal to zero vector it can be written that $\Delta u = u$.
The linearization is then done by calculation of the Jacobi matrix of the subsystem mentioned in equations 6.9 taking β and ψ as state variables.

$$\mathbf{J} = \begin{pmatrix} -\sin(\beta_e + \psi_e) \cdot v & -\sin(\beta_e + \psi_e) \cdot v \\ \cos(\beta_e + \psi_e) \cdot v & \cos(\beta_e + \psi_e) \cdot v \end{pmatrix} \tag{6.14}$$

where β_e and ψ_e are state values in which the system is linearized and v is the velocity at the reference point on the vehicle taken as a variable parameter. Matrices of the linear predictor could be then composed of the LPV model extended with heading integrator and the linearized velocity model from equation 6.14. The resulting matrices and the predictor can be seen in equations 6.15 - 6.17.

$$\underbrace{\begin{pmatrix} -\frac{C_f + C_r}{mv} & \frac{l_r C_r - l_f C_f}{mv^2} - 1 & 0 & 0 & 0 \\ \frac{l_r C_r - l_f C_f}{I_z} & -\frac{l_f^2 C_f + l_r^2 C_r}{v I_z} & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ -\sin(\beta_e + \psi_e) \cdot v & 0 & -\sin(\beta_e + \psi_e) \cdot v & 0 & 0 \\ \cos(\beta_e + \psi_e) \cdot v & 0 & \cos(\beta_e + \psi_e) \cdot v & 0 & 0 \end{pmatrix}}_{\mathbf{A}_p}, \tag{6.15}$$

$$\underbrace{\begin{pmatrix} \frac{C_f}{mv} & \frac{C_r}{mv} \\ \frac{l_f C_f}{I_z} & -\frac{l_r C_r}{I_z} \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{pmatrix}}_{\mathbf{B}_p}, \tag{6.16}$$

$$\begin{pmatrix} \Delta\dot{\beta} \\ \Delta\ddot{\psi} \\ \Delta\dot{\psi} \\ \Delta\dot{N} \\ \Delta\dot{E} \end{pmatrix} = \mathbf{A}_p \cdot \begin{pmatrix} \Delta\beta \\ \Delta\dot{\psi} \\ \Delta\psi \\ \Delta N \\ \Delta E \end{pmatrix} + \mathbf{B}_p \cdot \begin{pmatrix} \delta_f \\ \delta_r \end{pmatrix}. \tag{6.17}$$

Because the linearization is done along the trajectory it is necessary to calculate the vector $\dot{x}_p$ using the formula 6.18.

$$\dot{x}_p = \mathbf{A}_p \cdot x_p, \tag{6.18}$$

where $x_p = \begin{pmatrix} \beta_e & \dot{\psi}_e & \psi_e & N_e & E_e \end{pmatrix}^T$ is the operational point and $\dot{x}_p = \begin{pmatrix} \beta_d & \dot{\psi}_d & \psi_d & N_d & E_d \end{pmatrix}$ describes a trajectory the system's states are developing. The operational point was chosen as the position and heading of the shadow vehicle (see subsection 5.3) with zero the side-slip angle and yaw rate.

The predictor has to be discretized every time any of the parameters changes its value. The discretization method used for the discretization of the predictor was the zero-order hold method. The predictor was discretized with sample period $T_p = 0.1$ s. The discretization was done only for the subsystem

$$\Delta x = \mathbf{A}_p \Delta x + \mathbf{B}_p u, \tag{6.19}$$

because only this subsystem will be used as the predictor inside the MPC framework.
For the discretization, it was necessary to calculate the matrix of state development $e^{\mathbf{A}_p T}$. The system matrices were then discretized using formulas 6.20.

$$\mathbf{A}_d = e^{\mathbf{A}_p T_p} \tag{6.20}$$

$$\mathbf{B}_d = \mathbf{B}_p \int_0^{T_p} e^{\mathbf{A}_p \tau} d\tau \tag{6.21}$$

6.2.2 Prediction reference

Path tracking algorithm using MPC needs to know future references to be able to fully exploit its predictive nature. The reference signals have to be evenly spaced in time according to the sample period of the predictor T_p. Shadow vehicle concept introduced in 5.3 could be used to project the position of the vehicle on the path and then "drive" the shadow vehicle along the path and "sample" points on the path to get the reference signals which can be seen in figure 6.4.

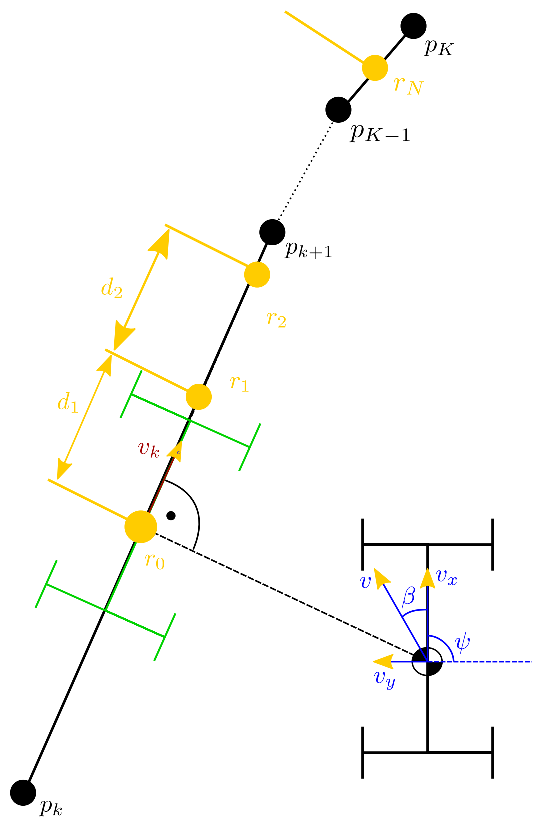


Figure 6.4: Visualization of prediction reference

Assuming that the vehicle is moving with velocity $|v|$ there exists a projection of the velocity vector v on the path. This is the velocity vector of the shadow vehicle v_s and it describes how fast the vehicle is moving along the path. The size of the shadow vehicle velocity vector $|v_s|$ is used to calculate spacing between reference points on the path. The spacing is calculated using formula 6.22

$$d = T_p \cdot |v_s|, \tag{6.22}$$

where T_p is the sampling period of the predictor, and $|v_s|$ is the size of the velocity vector of the shadow vehicle when driving along the path segment described by points p_k and p_{k+1}. The first point s_0 is the shadow vehicle position. Then there can be constructed points s_l, $l \in 1, \dots H$ so that they lie on the path and their distance along the path is equal to d.

Reference signal $r_l = \begin{pmatrix} \psi_l & N_l & E_l \end{pmatrix}^T$ then consists of north and east coordinates, which are directly taken from coordinates of points s_l, $1, \dots H$ on the path, and the vehicle body heading ψ_l, which is taken from initial point p_k of the line segment to which s_l belongs.

Due to the formulation of MPC problem described in section 6.2.1 the references have to be further transformed into a suitable form. In order to do that it is necessary to introduce a new vector $r_d = \begin{pmatrix} \psi_d & N_d & E_d \end{pmatrix}^T$ where N_d, E_d

and ψ_d are elements of vector $\dot{x}_p$ defined in section 6.2.1. The transformed reference vector can be then described by equation 6.23.

$$\hat{r}_j = r_j - j \cdot T_p \cdot r_d,\ j \in 1, \dots H \tag{6.23}$$

where r_j is the transformed reference vector and T_p is the predictor sampling period.

6.2.3 Formulation of optimization problem

The tracking problem described in 6.2.1 can be written as an optimization problem. There is introduced new labeling where state increases are labeled $z_k = x_k - x_p = \Delta x$ and control input is labeled $v_k = u_k$. For the sake of simplicity, it can be assumed that in every sampling period T_p there is a new measurement z_0 and the MPC algorithm is reinitialized. The optimization problem is then described by the equations below.

$$\min_{z_1,\dots,z_N,v_0,\dots,v_{N-1}\in\mathbb{R}^m} \quad J(z_1,\dots,z_N,v_0,\dots,v_{N-1},\hat{r}_1,\dots,\hat{r}_N) \tag{6.24}$$

$$s.t.: \quad z_k = \mathbf{A}z_{k-1} + \mathbf{B}v_{k-1},\ k \in 1,\dots,N \tag{6.25}$$

with cost function defined as

$$J(z_1,\dots,z_N,v_0,\dots,v_{N-1}) = \tag{6.26}$$

$$= \sum_{k=1}^{N} \frac{1}{2}((\hat{r}_k - \mathbf{C}z_k)^T \mathbf{Q}(\hat{r}_k - \mathbf{C}z_k) + v_{k-1}^T \mathbf{R} v_{k-1}). \tag{6.27}$$

This formulation has a problem in that the steady-state error won't be zero because the controller minimizes absolute values of inputs u_k, thus controller cannot maintain the constant non-zero value of inputs v_k and cannot regulate error to zero in steady state.
This tracking problem can be solved by the augmentation of the controlled system. The system will be augmented by introducing additional state equations 6.28.

$$\Delta v_k = v_k - v_{k-1} \tag{6.28}$$

$$v_k = v_{k-1} + \Delta v_k \tag{6.29}$$

This means that there is a new state $v_{k-1} = z_k^u$ and Δv is virtual input. State equation can be then rewritten into the following vector equations

$$\tilde{z}_{k+1} = \underbrace{\begin{pmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{0} & \mathbf{I} \end{pmatrix}}_{\tilde{\mathbf{A}}} \underbrace{\begin{pmatrix} z_k \\ z_k^u \end{pmatrix}}_{\tilde{z}_k} + \underbrace{\begin{pmatrix} \mathbf{B} \\ \mathbf{I} \end{pmatrix}}_{\tilde{\mathbf{B}}} \Delta v_k, \tag{6.30}$$

$$y_k = \underbrace{\begin{pmatrix} \mathbf{C} & \mathbf{0} \end{pmatrix}}_{\tilde{\mathbf{C}}} \begin{pmatrix} z_k \\ z_k^u \end{pmatrix} \tag{6.31}$$

Cost function J can be now reformulated using augmented system matrices into

$$J = \sum_{k=1}^{N} \frac{1}{2}((\hat{r}_k - \tilde{\mathbf{C}}\tilde{z}_k)^T \mathbf{Q}(\hat{r}_k - \tilde{\mathbf{C}}\tilde{z}_k) + \Delta v_{k-1}^T \mathbf{R} \Delta v_{k-1}). \tag{6.32}$$

The cost function J can be further modified and after modification and removal of constant terms, which have no effect on minimization, there remains the following cost function

$$J = \sum_{k=1}^{N} (\frac{1}{2}\tilde{z}_k{}^T \tilde{\mathbf{C}}^T \mathbf{Q}\tilde{\mathbf{C}}\tilde{z}_k - \hat{r}_k^T \mathbf{Q}\tilde{\mathbf{C}}\tilde{z}_k + \frac{1}{2}\Delta v_{k-1}^T \mathbf{R}\Delta v_{k-1}). \tag{6.33}$$

The optimization variables can be further associated into vectors

$$\tilde{z} = \begin{pmatrix} \tilde{z}_1 \\ \tilde{z}_2 \\ \vdots \\ \tilde{z}_N \end{pmatrix}, \ \Delta v = \begin{pmatrix} \Delta v_0 \\ \Delta v_1 \\ \vdots \\ \Delta v_{N-1} \end{pmatrix}, \ r = \begin{pmatrix} \hat{r}_1 \\ \hat{r}_2 \\ \vdots \\ \hat{r}_N \end{pmatrix}. \tag{6.34}$$

The cost function can be then written as a vector function of these associated optimization variables

$$\begin{aligned} J = \frac{1}{2}\tilde{z}^T \underbrace{\begin{pmatrix} \tilde{\mathbf{C}}^T\mathbf{Q}\tilde{\mathbf{C}} & & \\ & \ddots & \\ & & \tilde{\mathbf{C}}^T\mathbf{Q}\tilde{\mathbf{C}} \end{pmatrix}}_{\bar{\bar{\mathbf{Q}}}} \tilde{z} - \hat{r} \underbrace{\begin{pmatrix} \mathbf{Q}\tilde{\mathbf{C}} & & \\ & \ddots & \\ & & \mathbf{Q}\tilde{\mathbf{C}} \end{pmatrix}}_{\bar{\bar{\mathbf{T}}}} \tilde{z} \\ + \Delta v \underbrace{\begin{pmatrix} \mathbf{R} & & \\ & \ddots & \\ & & \mathbf{R} \end{pmatrix}}_{\bar{\bar{\mathbf{R}}}} \Delta v \end{aligned}. \tag{6.35}$$

Minimization problem constraints can be written similarly as

$$\tilde{z} = \begin{pmatrix} \mathbf{0} & & & \\ \tilde{\mathbf{A}} & \ddots & & \\ & \ddots & \ddots & \\ & & \tilde{\mathbf{A}} & \mathbf{0} \end{pmatrix} \tilde{z} + \begin{pmatrix} \tilde{\mathbf{B}} & & \\ & \ddots & \\ & & \tilde{\mathbf{B}} \end{pmatrix} \Delta v + \begin{pmatrix} \tilde{\mathbf{A}} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \end{pmatrix} \tilde{z}_0. \tag{6.36}$$

The number of optimization variables can be further reduced by converting this problem to a so-called dense formulation by elimination of state variables using only initial state $\tilde{z}_0$ and vector of optimization variables Δv. All states can be expressed as the following function of initial state z_0 and optimization variables vector Δv described as

$$\tilde{z} = \underbrace{\begin{pmatrix} \tilde{\mathbf{A}} \\ \tilde{\mathbf{A}}^2 \\ \vdots \\ \tilde{\mathbf{A}}^k \\ \vdots \\ \tilde{\mathbf{A}}^N \end{pmatrix}}_{\overline{\overline{\mathbf{C}}}} \tilde{z}_0 + \underbrace{\begin{pmatrix} \tilde{\mathbf{B}} & & & & \\ \tilde{\mathbf{A}}\tilde{\mathbf{B}} & \tilde{\mathbf{B}} & & & \\ \vdots & & \ddots & & \\ \tilde{\mathbf{A}}^{k-1}\tilde{\mathbf{B}} & \tilde{\mathbf{A}}^{k-2}\tilde{\mathbf{B}} & \dots & \tilde{\mathbf{B}} & \\ \vdots & & & \ddots & \\ \tilde{\mathbf{A}}^{N-1}\tilde{\mathbf{B}} & \dots & & \dots & \tilde{\mathbf{B}} \end{pmatrix}}_{\hat{\mathbf{A}}} \Delta v. \tag{6.37}$$

the cost function can be then reformulated into the form only with vector Δv as optimization variable and furthermore into the form of QP problem cost function as can be seen in equations 6.38.

$$J = \frac{1}{2}\Delta v^T \underbrace{(\hat{\mathbf{A}}^T\overline{\overline{\mathbf{Q}}}\hat{\mathbf{A}} + \overline{\overline{\mathbf{R}}})}_{\mathbf{H}} \Delta v + \begin{pmatrix} \tilde{x}_0^T & r^T \end{pmatrix} \underbrace{\begin{pmatrix} \overline{\overline{\mathbf{CQ}}}\hat{\mathbf{A}} \\ -\overline{\overline{\mathbf{T}}}\hat{\mathbf{A}} \end{pmatrix}}_{\mathbf{F}} \Delta u \tag{6.38}$$

$$J = \frac{1}{2}\Delta u^T \mathbf{H} \Delta v + \underbrace{\begin{pmatrix} \tilde{x}_0^T & r^T \end{pmatrix} \mathbf{F}}_{\mathbf{b}} \Delta v \tag{6.39}$$

$$J = \frac{1}{2}\Delta u^T \mathbf{H} \Delta v + \mathbf{b}\Delta v \tag{6.40}$$

Minimization of this cost function then yields a sequence of control input increments. This incremental formulation of the MPC problem gives integral behavior of the controller because of the minimization of input increments instead of absolute values of inputs.
Minimization of cost function J gives a whole vector of control inputs over the given horizon and thus calculation of current control input v_t is needed. Current control input can be obtained by using element Δv_0 from vector Δv and last applied control input v_{t-1} in a way described in equation 6.41.

$$v_t = v_{t-1} + \Delta v_0 \tag{6.41}$$

This minimization solves only unconstrained problems. The vehicle has constraints on its steering angle thus there have to be added equations describing bounds on control inputs. Let's assume that v_{min} is a vector of lower bounds of control inputs and v_{max} is a vector of upper bounds of control inputs. Input constraints can be then written as

$$v_{min} \leq v_{k-1} \leq v_{max},\ k = 1, \dots, N. \tag{6.42}$$

This equation can be then modified using system augmentation from 6.30 and selection matrix $\mathbf{E}$ to

$$v_{min} \leq \tilde{\mathbf{A}}\tilde{z}_{k-1} + \tilde{\mathbf{B}}\Delta v_{k-1} \leq v_{max},\ k = 1, \dots, N. \tag{6.43}$$

Selection matrix $\mathbf{E}$ selects states of the augmented system, which will be constrained. When only inputs are to be constrained the selection matrix $\mathbf{E}$ has the following form

$$\mathbf{E} = \begin{pmatrix} \mathbf{0} & \mathbf{I} \end{pmatrix}, \tag{6.44}$$

where matrix $\mathbf{I} \in \mathbb{R}^{m \times m}$ is identity matrix and m is number of controlled inputs.
Matrices from 6.37 can be then used to eliminate state vector z_k from the equation and to adjust the constraint equation for whole optimization sequence Δv as follows

$$\underbrace{\begin{pmatrix} \mathbf{I} \\ \vdots \\ \mathbf{I} \end{pmatrix}}_{\bar{\mathbf{I}}} v_{min} \leq \underbrace{\begin{pmatrix} \mathbf{E} & & \\ & \ddots & \\ & & \mathbf{E} \end{pmatrix}}_{\bar{\mathbf{E}}} (\hat{\mathbf{A}}\Delta v + \overline{\overline{\mathbf{C}}}\tilde{z}_0) \leq \underbrace{\begin{pmatrix} \mathbf{I} \\ \vdots \\ \mathbf{I} \end{pmatrix}}_{\bar{\mathbf{I}}} v_{max} \tag{6.45}$$

Equation adjustment then give

$$\underbrace{\bar{\mathbf{I}}u_{min} - \bar{\mathbf{E}}\overline{\overline{\mathbf{C}}}\tilde{x}_t}_{b_l} \leq \bar{\mathbf{E}}\hat{\mathbf{A}}\Delta u \leq \underbrace{\bar{\mathbf{I}}u_{max} - \bar{\mathbf{E}}\overline{\overline{\mathbf{C}}}\tilde{x}_t}_{b_u}, \tag{6.46}$$

which describes constraint on the sequence of input increments Δv.

The whole optimization problem can be then described as

$$\min_{\Delta v} \quad \tfrac{1}{2}\Delta v^T \mathbf{H} \Delta v + \mathbf{b}\Delta v^T, \tag{6.47}$$

$$s.t.: \quad b_l \leq \bar{\mathbf{E}}\hat{\mathbf{A}}\Delta u \leq b_u. \tag{6.48}$$

6.2.4 Implementation

This algorithm was at first implemented in Matlab & Simulink environment. This implementation allowed fast prototyping and debugging of the whole algorithm. The Simulink model implementing MPC framework was after testing and debugging used to generate a C++ class using the Embedded Coder [Matb] from Matlab & Simulink. This C++ class was then integrated into the ROS2 interface node to be able to communicate with the rest of the vehicle.

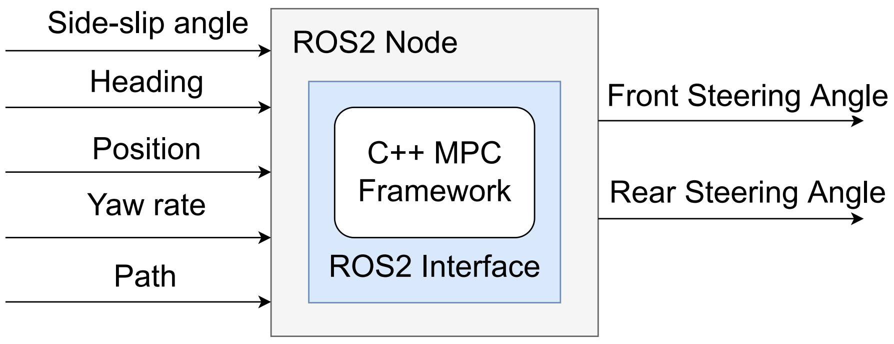


Figure 6.5: Block Diagram of MPC Node

The solver used to solve the QP problem defined in the section 6.2.3 was *quadprog* [Matc] from Matlab & Simulink. This solver is capable of solving constrained QP problems. The solver takes the QP problem in from described by equation 6.49.

$$\min_x \frac{1}{2} x^T \mathbf{H} x + f^T x, \tag{6.49}$$

$$s.t. : \mathbf{A} \cdot x \leq b, \tag{6.50}$$

$$\mathbf{A}_{eq} \cdot x = b_{eq} \tag{6.51}$$

$$b_l \leq x \leq b_u \tag{6.52}$$

Chapter 7

Experiments

The path algorithms were tested in two scenarios. The first scenario was the piecewise linear path with two turns. This scenario is good for testing because the two curves are curved in the opposite direction.
The second scenario was the big loop around the testing facility this scenario was designed to test if the algorithms can sustain longer runs.
The velocity controller was tested on a straight road with different velocity references.

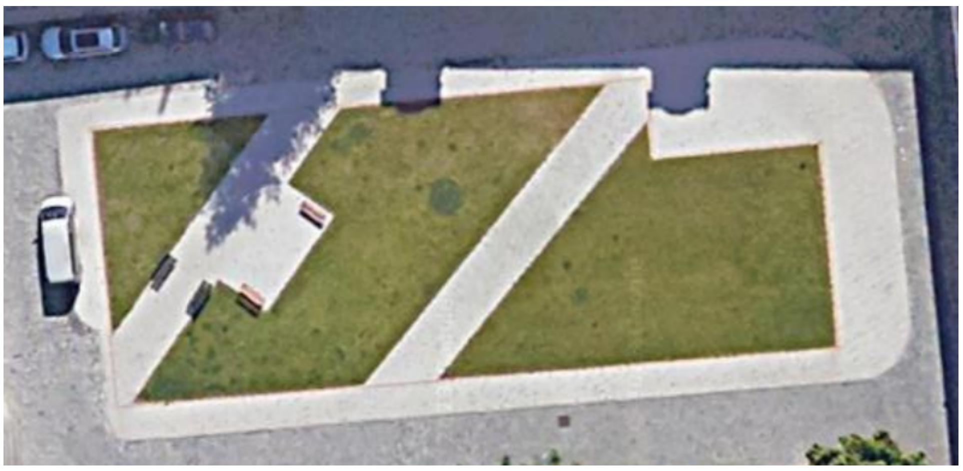


Figure 7.1: Bird's eye view of the testing facility

7.1 Velocity Controller

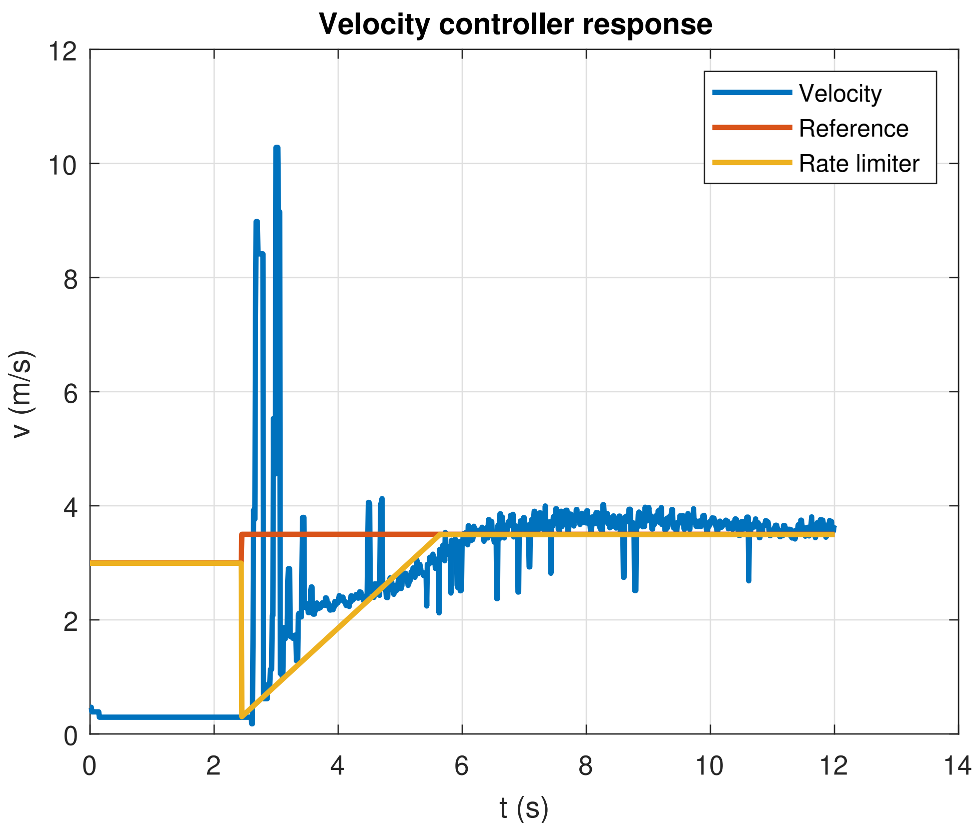


Figure 7.2: Comparison of measurements and reference signals

The function of the velocity controller can be seen in figure 7.2. Approximately 2 s from the start of the experiment a new reference signal arrives. The rate limiter algorithm is initialized to the current velocity of the vehicle and starts to rise with a slope of $a_{lim} = 1\text{m/s}$. There are also large oscillations of the vehicle velocity visible at the start of the experiment. This effect is probably caused by deadzone of the motor controller and also the measurement of the velocity from revolutions of the front wheels. On the other hand, after the deadzone region is surpassed the vehicle tends to the reference velocity with a small overshoot.

7.2 Stanley Control Law

7.2.1 Controller without predictive measurements

The tuning constants used in these testing scenarios can be seen in table 7.1

k_{soft}	1
k	0.5
k_{yaw}	1

Table 7.1: Stanley Control Law tuning constants

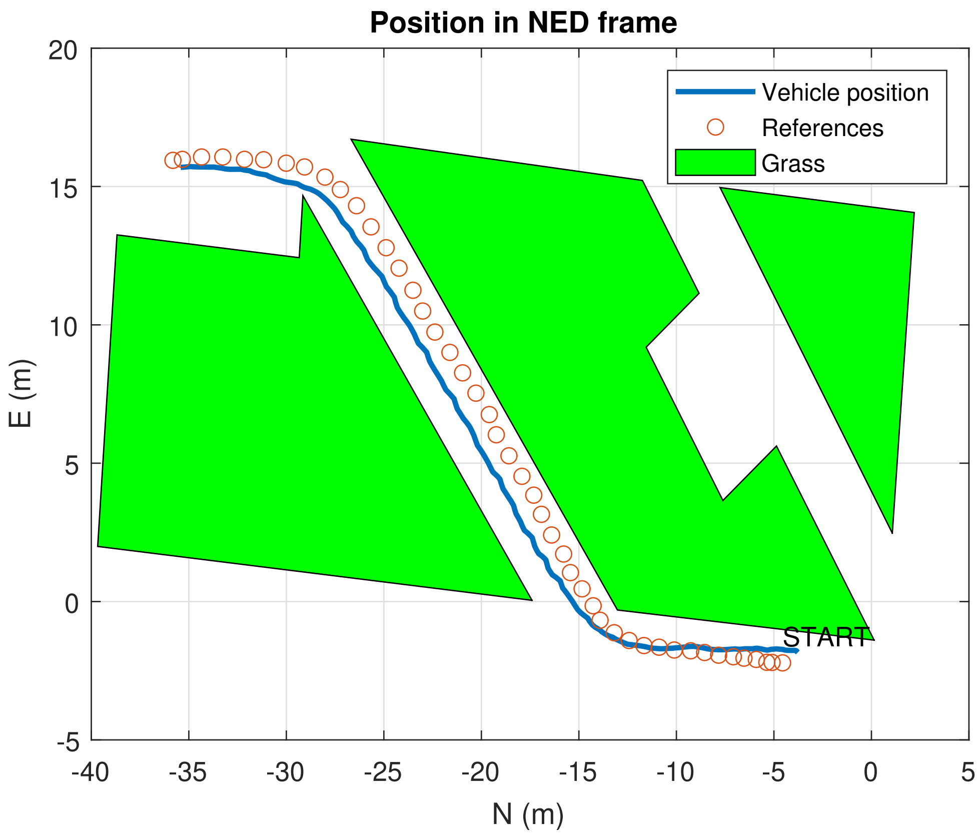


Figure 7.3: Vehicle positions and references in NE plane

Figure 7.3 shows the measured position of the vehicle compared to the reference path. From picture, it can be seen that the Stanley Control Law can track the path correctly but with small deviation. This deviation results from its non-predictive nature and also from the nonideal vehicle platform. The platform has got some asymmetry in its steering geometry so it spontaneously turns a bit to the right. Stanley Control Law can't solve this problem by itself because the control law is equivalent to the proportional controller.

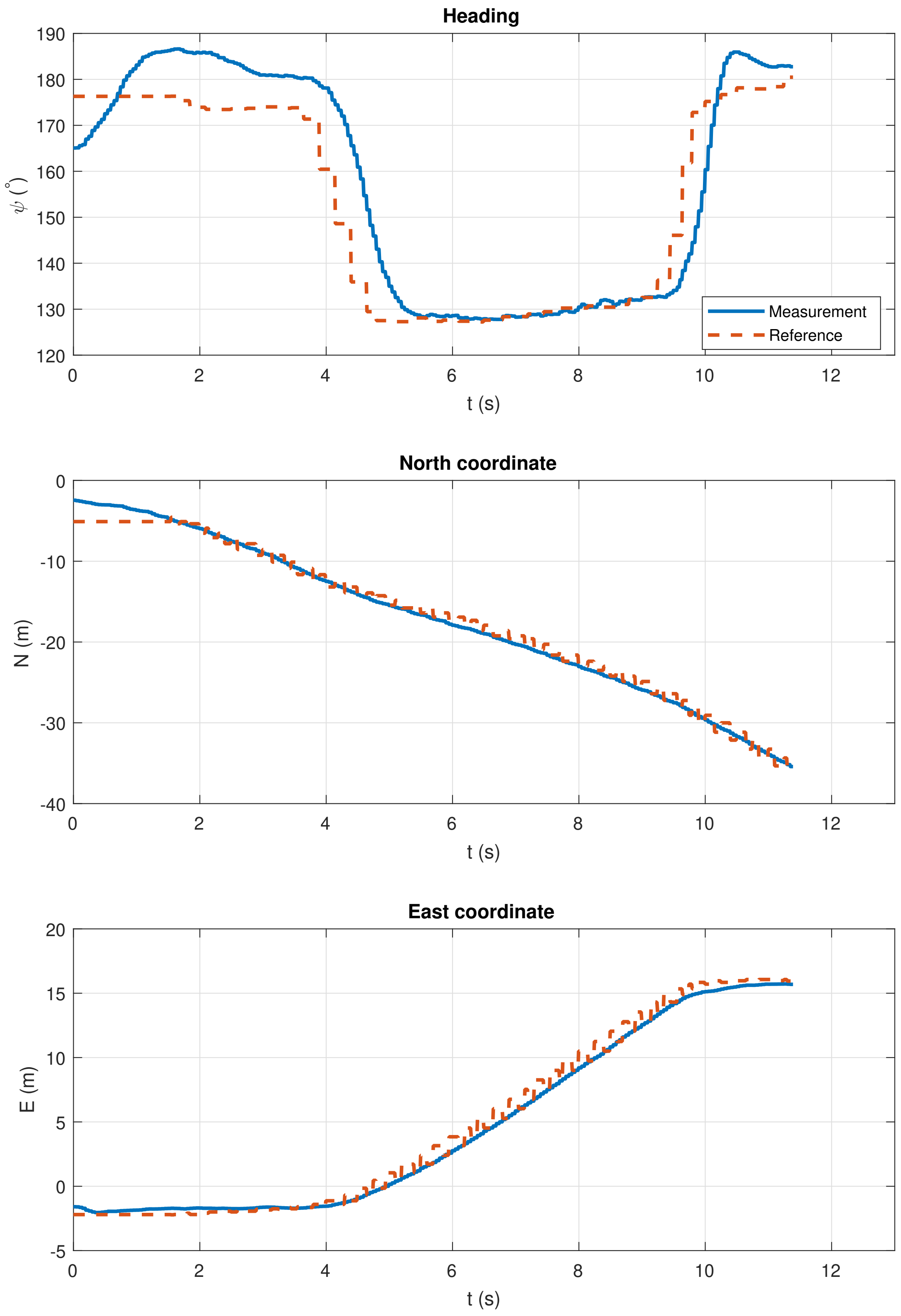


Figure 7.4: Comparison of measurements and reference signals

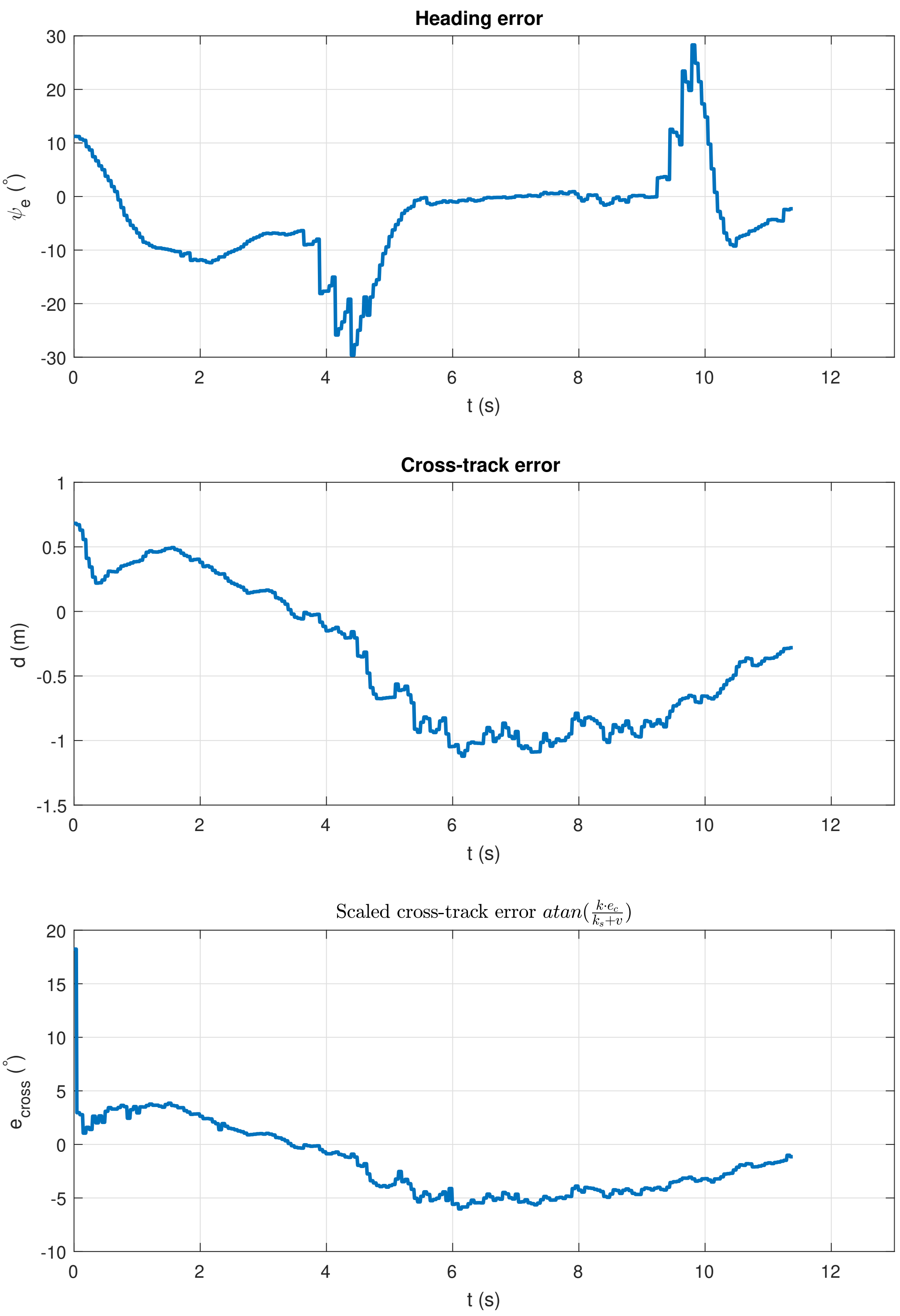


Figure 7.5: Control errors

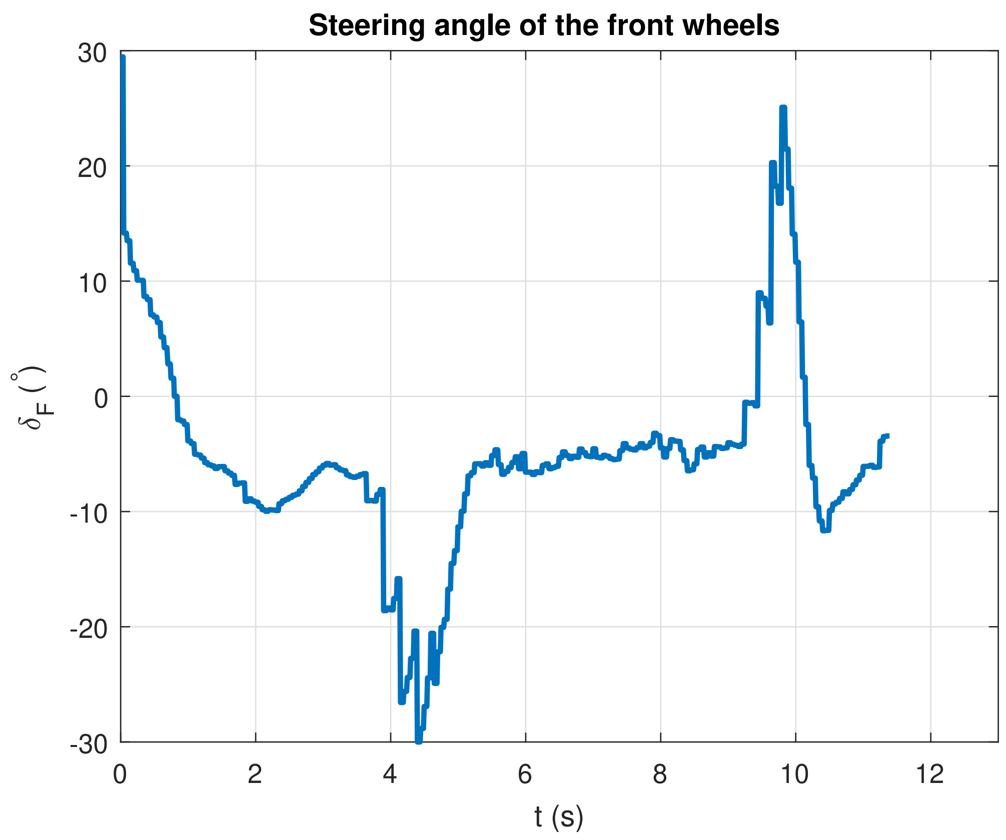


Figure 7.6: Velocity profile

Figure 7.6 shows that the control action on the front steering angle isn't smooth. This effect is caused by the piece-wise continuous heading reference. The controller gets the reference and tries to track it and after the new reference is received there is a step from the previous reference and this results in the steps in the control action.

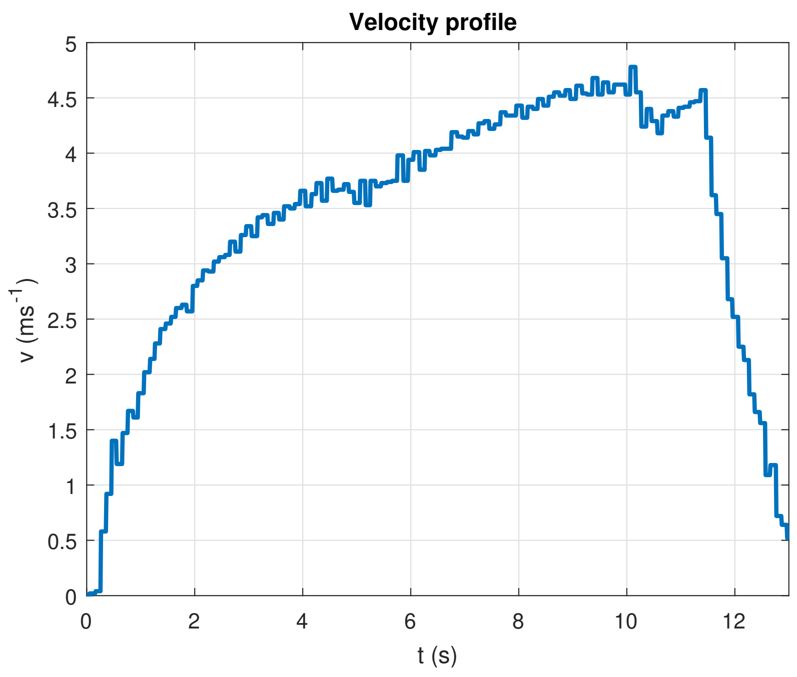


Figure 7.7: Vehicle positions and with references in NED plane

7.2.2 Controller with predictive measurements

The tuning constants used in this testing scenarios can be seen in table 7.2

k_{soft}	1
k	0.5
k_{yaw}	0.7
k_{lh}	0.3
t_{GAP}	0.3 s
d_0	0.5 m

Table 7.2: Predictive Stanley Control Law tuning constants

Short test

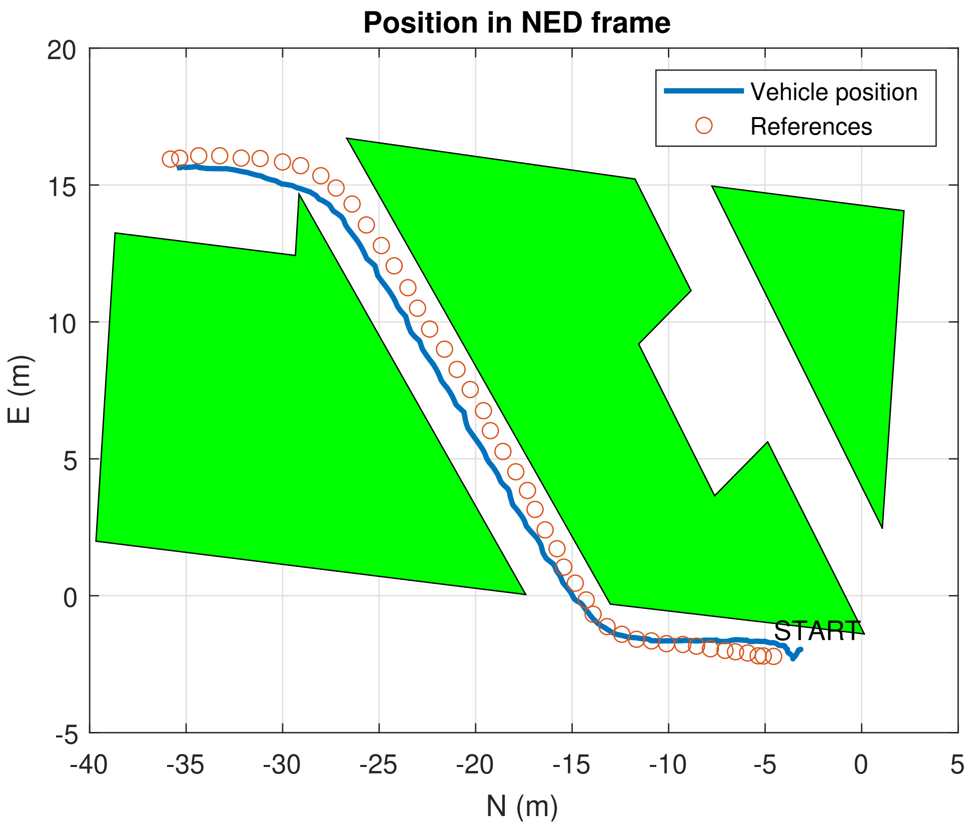


Figure 7.8: Vehicle positions and reference signals in NE plane

Figure 7.8 shows that the Stanley control law with a predictive part performed really well in the first turn because it started to turn sooner than thanks to the lookahead measurement. However, in the narrow part of the path it started to deviate from the path due to the spontaneous turn of the vehicle. The maximal deviation was in the final turn probably because the direction of the turn matches the direction of the spontaneous turn of the vehicle.

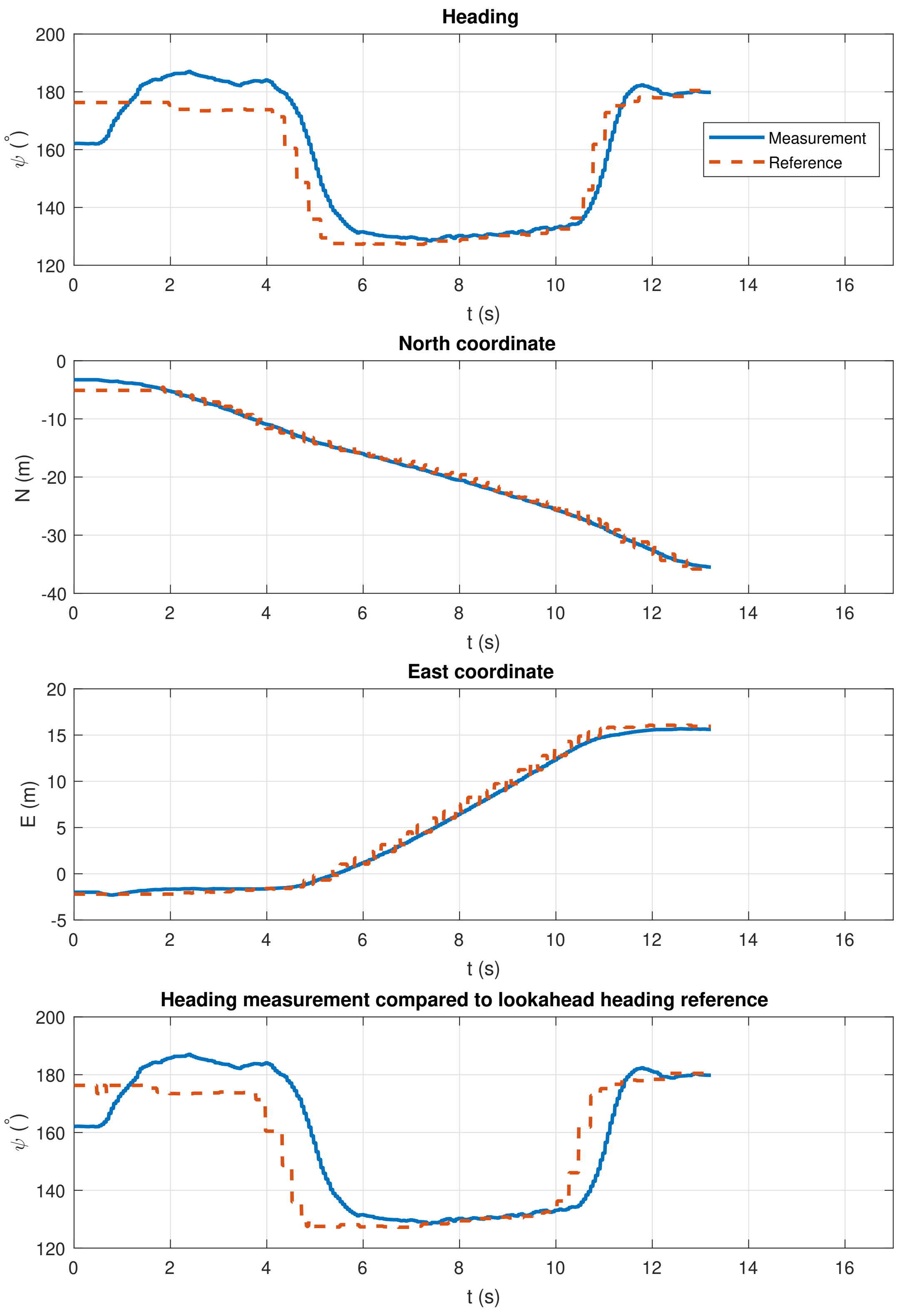


Figure 7.9: Comparison of measurements and reference signals

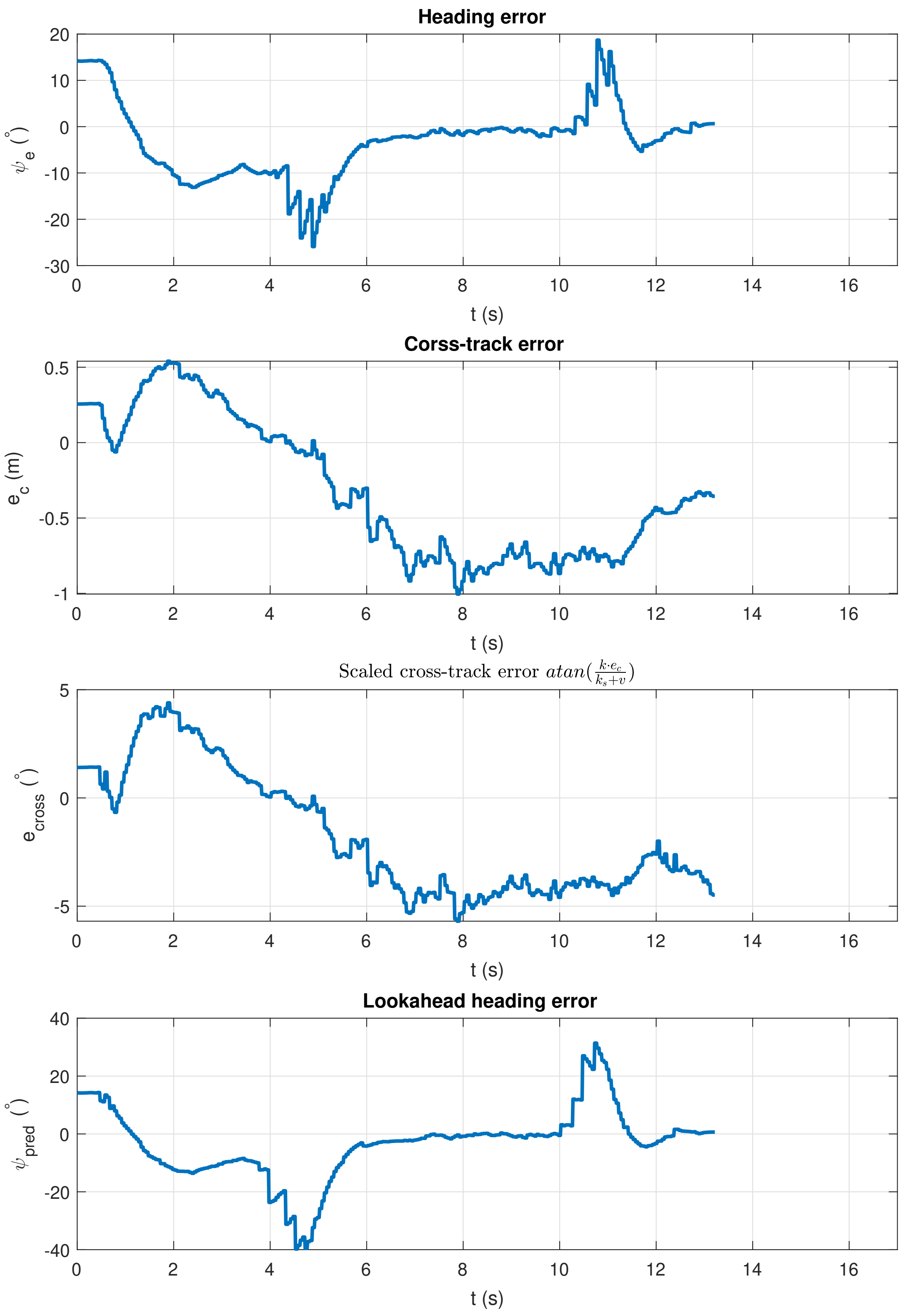


Figure 7.10: Control errors

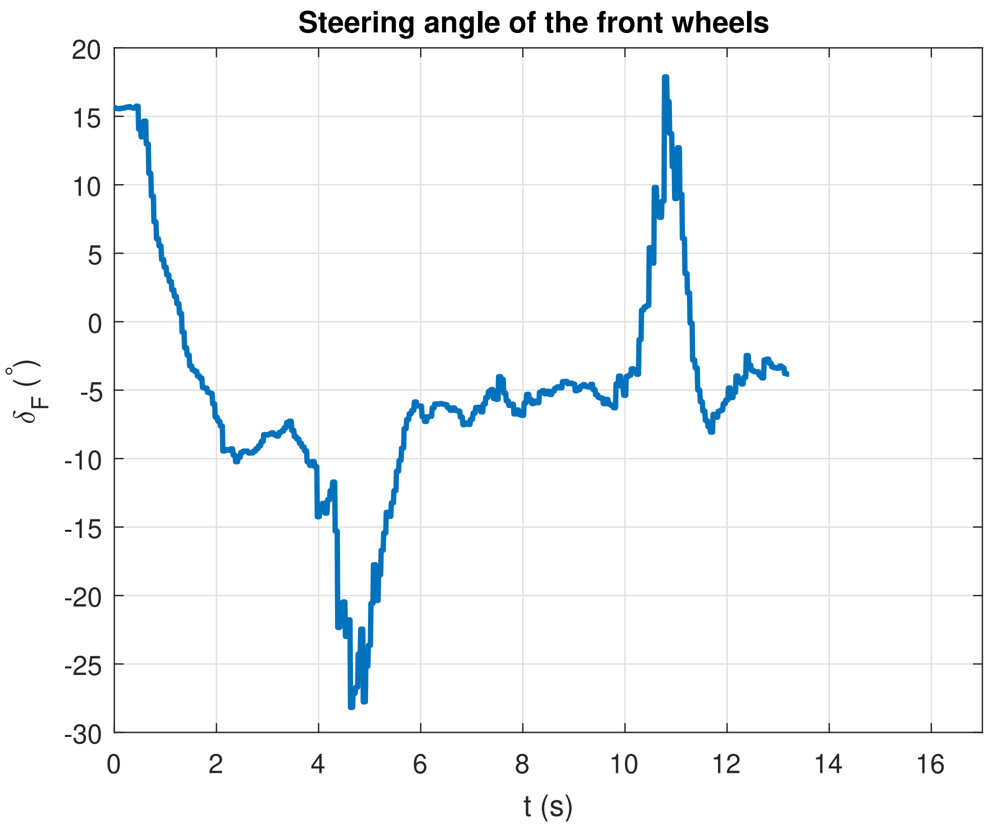


Figure 7.11: Control signal of the algorithm

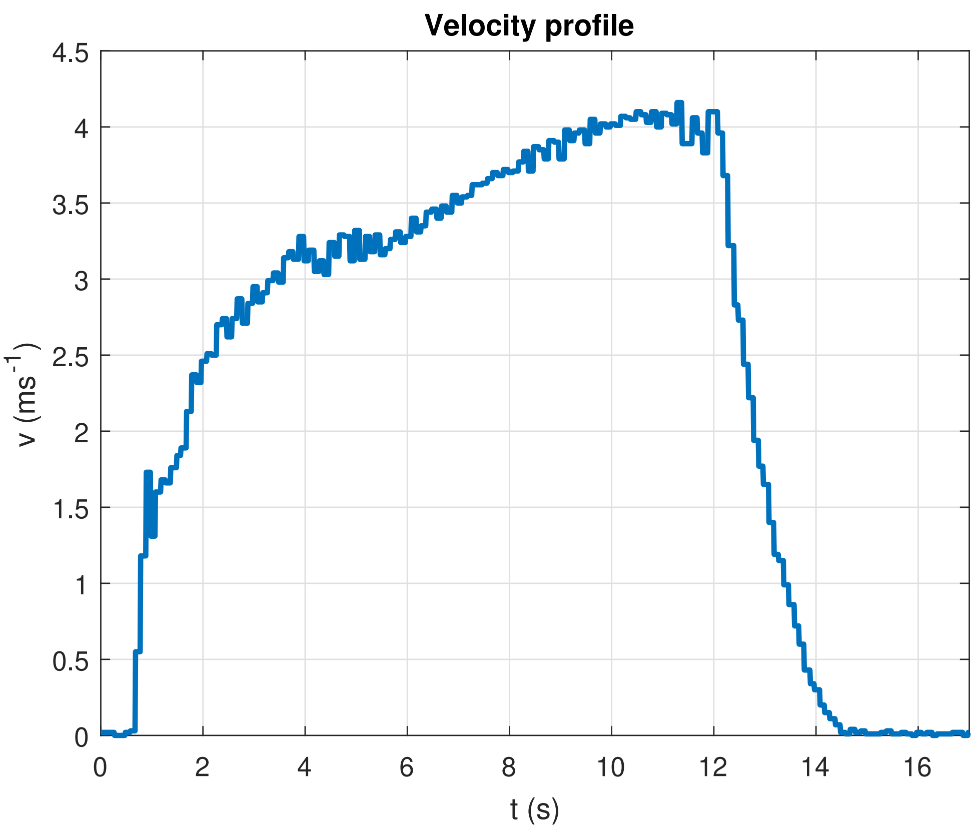


Figure 7.12: Predictive Stanley Control Law velocity profile

7.3 MPC based algorithm

MPC algorithm was tested in two different variants. The first variant of the MPC algorithm had steering with both front and rear axles enabled. The second variant had the rear axle steering disabled.

7.3.1 MPC with both axles steered

The cost matrices of this MPC variant can be seen below.

$$\mathbf{Q} = \begin{pmatrix} 100000 & 0 & 0 \\ 0 & 100 & 0 \\ 0 & 0 & 100 \end{pmatrix}, \tag{7.1}$$

$$\mathbf{R} = \begin{pmatrix} 1000 & 0 \\ 0 & 1000 \end{pmatrix}, \tag{7.2}$$

The prediction horizon of the MPC was set to $H_p = 10$.

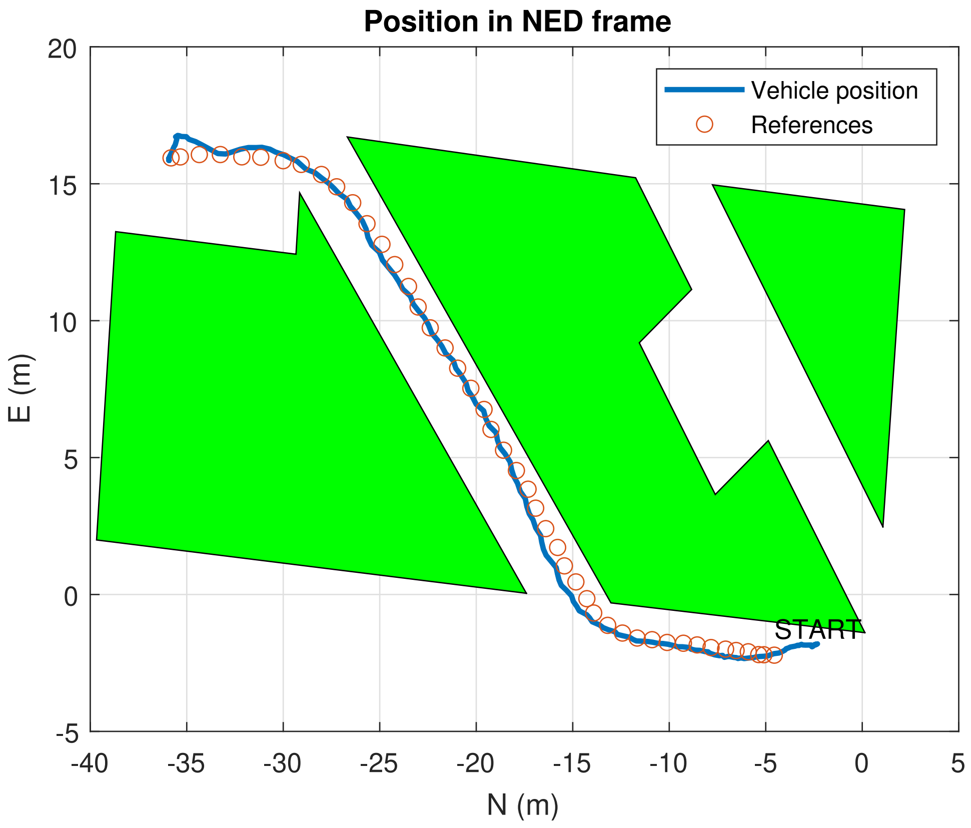


Figure 7.13: Vehicle positions and reference signals in NE plane

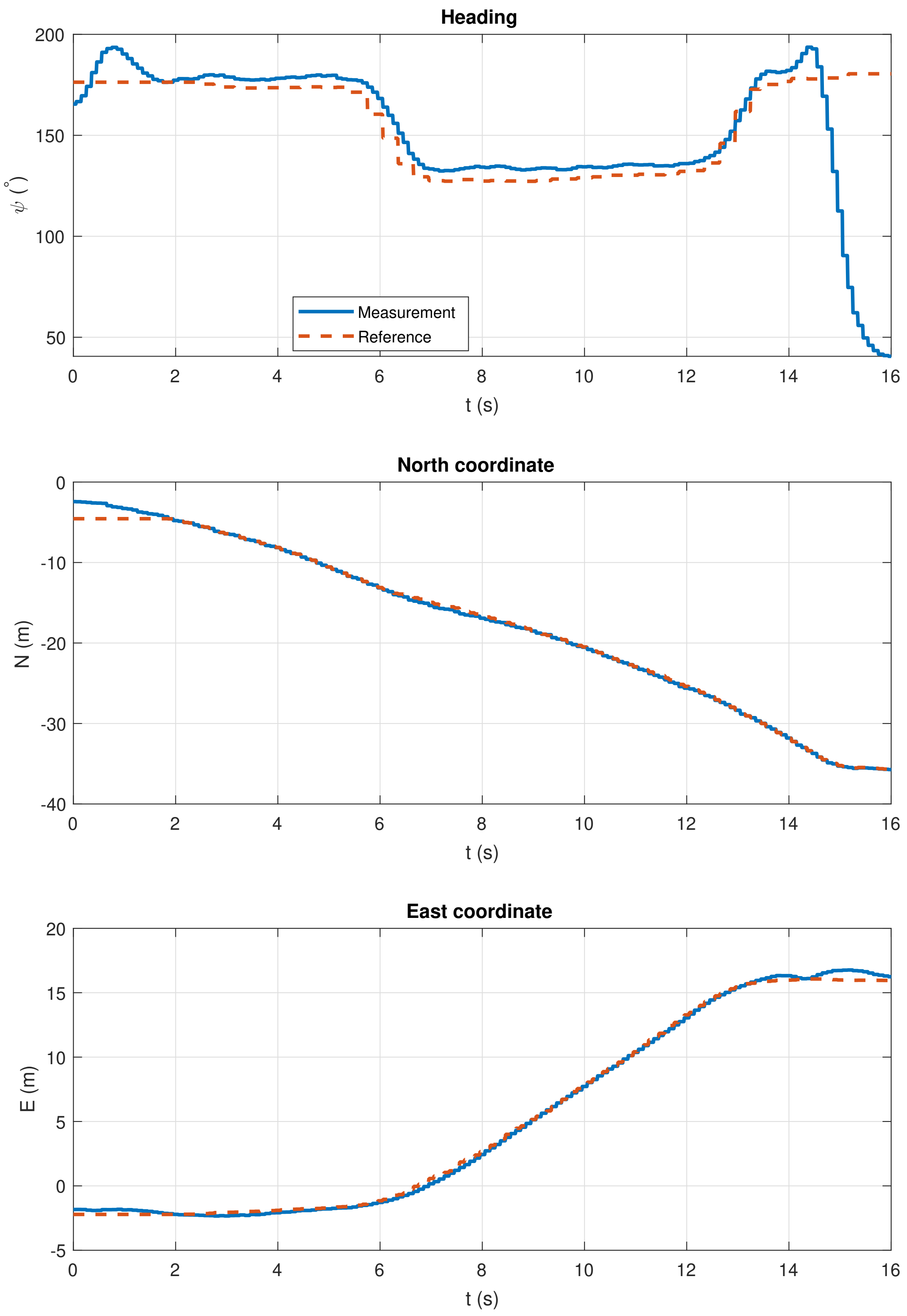


Figure 7.14: Measurement and reference signals comparison

This experiment shows the capability of MPC based control algorithm fully exploits the features of the over-actuated vehicle platform. Figure 7.17 that the vehicle is using the so-called crabwalk to track the reference position. The integral nature resulting from the difference formulation of the MPC is able to neglect the steady state error resulting from the nonideal steering geometry of the vehicle.

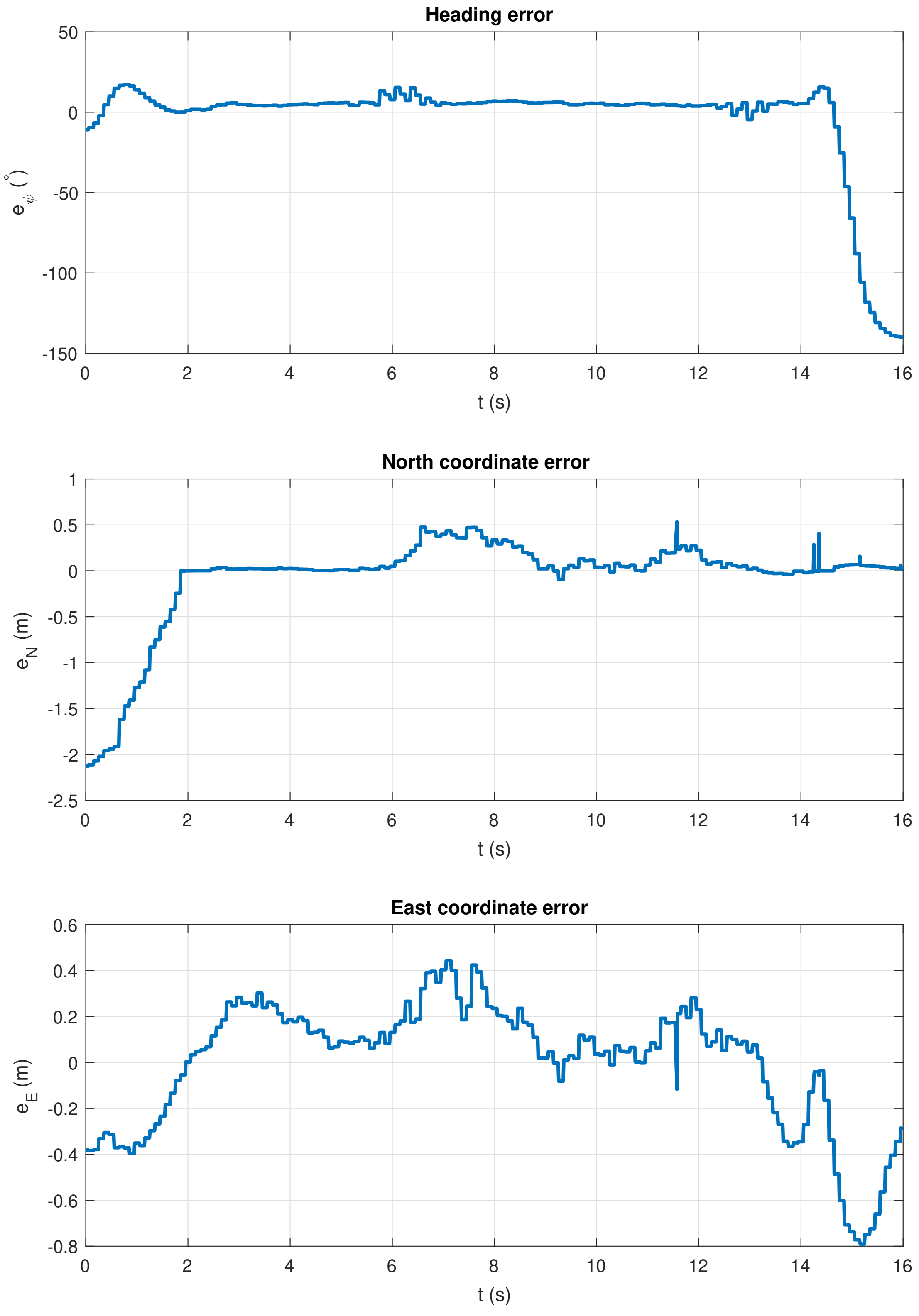
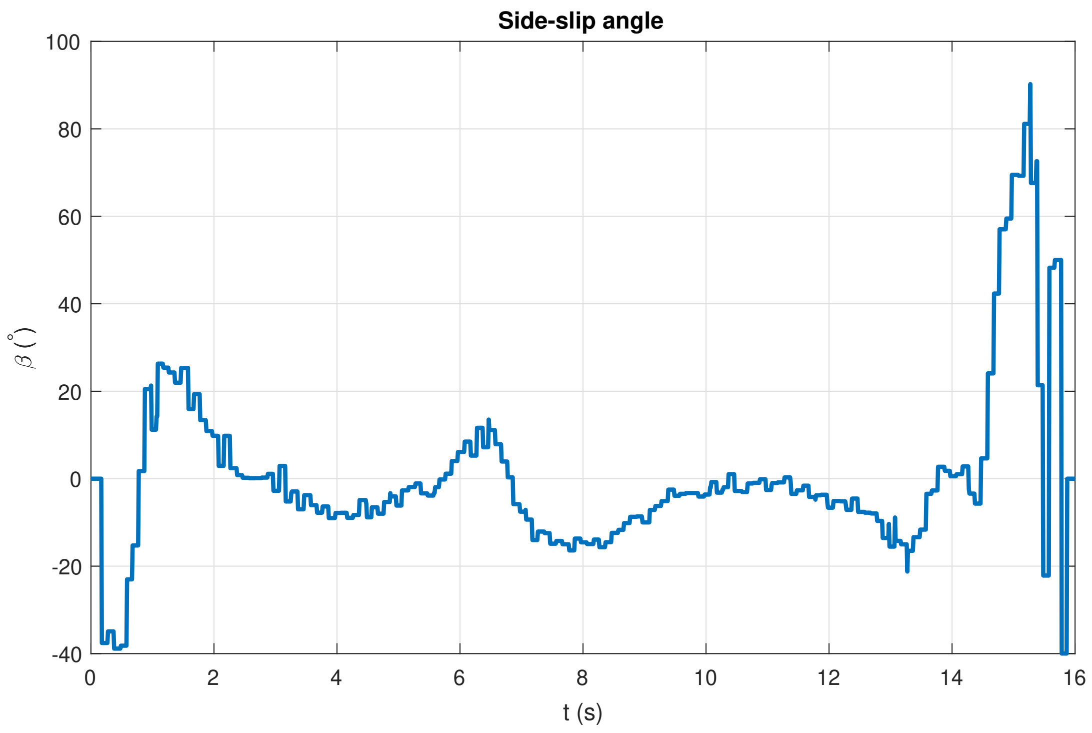


Figure 7.15: Control errors



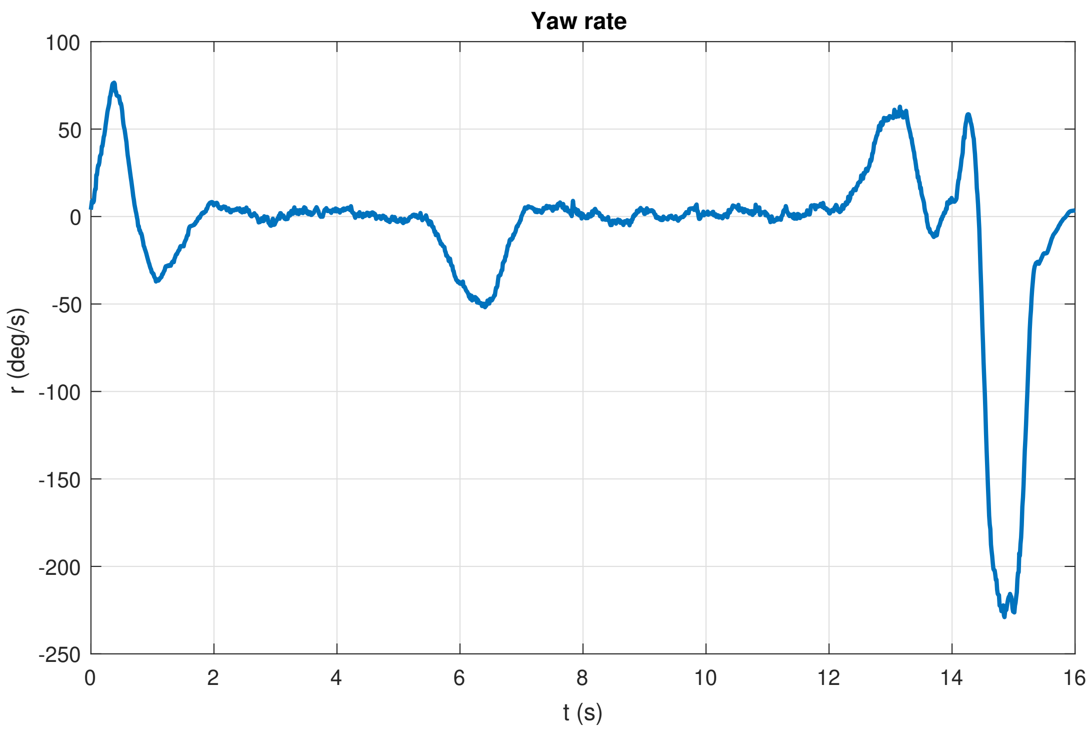


Figure 7.16: Untracked states of the system

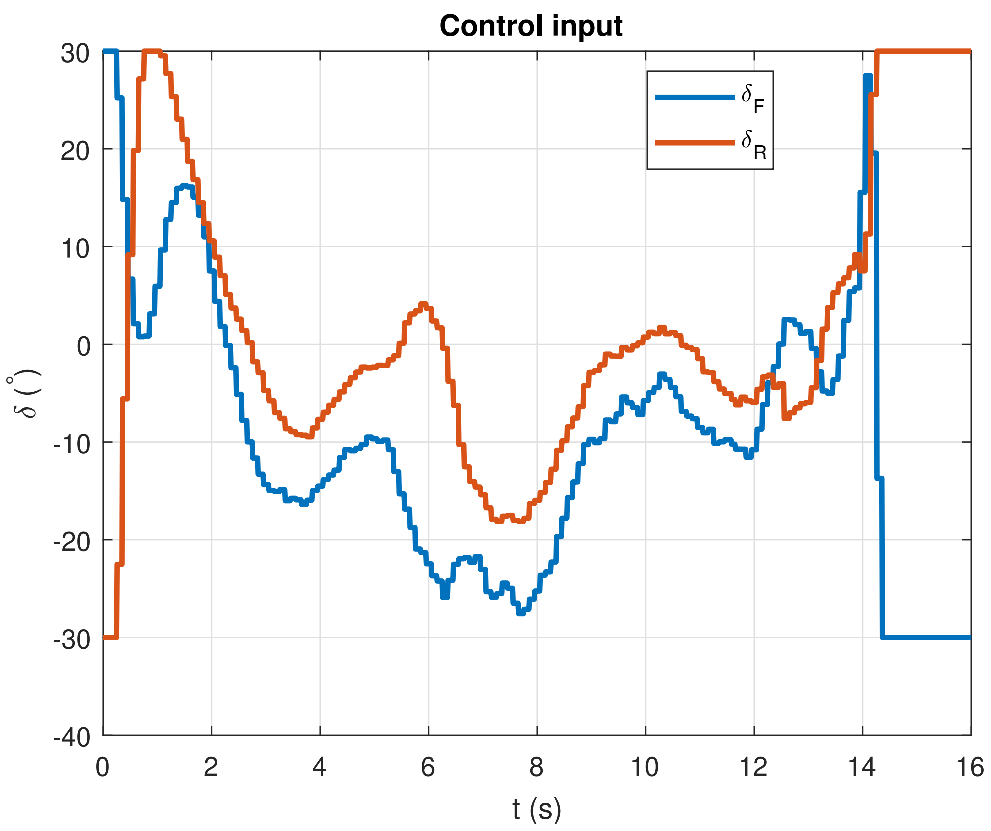


Figure 7.17: Control action of the algorithm

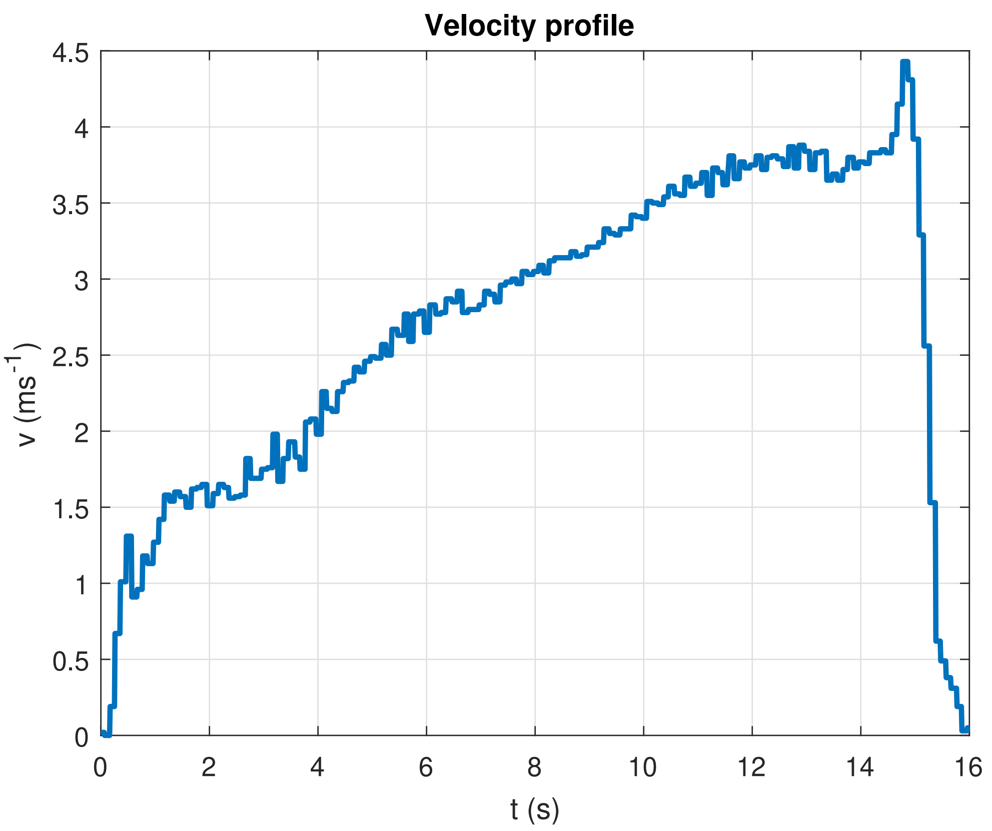


Figure 7.18: Velocity profile

7.3.2 MPC with the rear steering disabled

The cost matrices of this MPC variant can be seen below.

$$\mathbf{Q} = \begin{pmatrix} 1000 & 0 & 0 \\ 0 & 1000 & 0 \\ 0 & 0 & 1000 \end{pmatrix}, \tag{7.3}$$

$$\mathbf{R} = \begin{pmatrix} 1000 & 0 \\ 0 & 1000 \end{pmatrix}, \tag{7.4}$$

The prediction horizon of the MPC was set to $H_p = 10$.

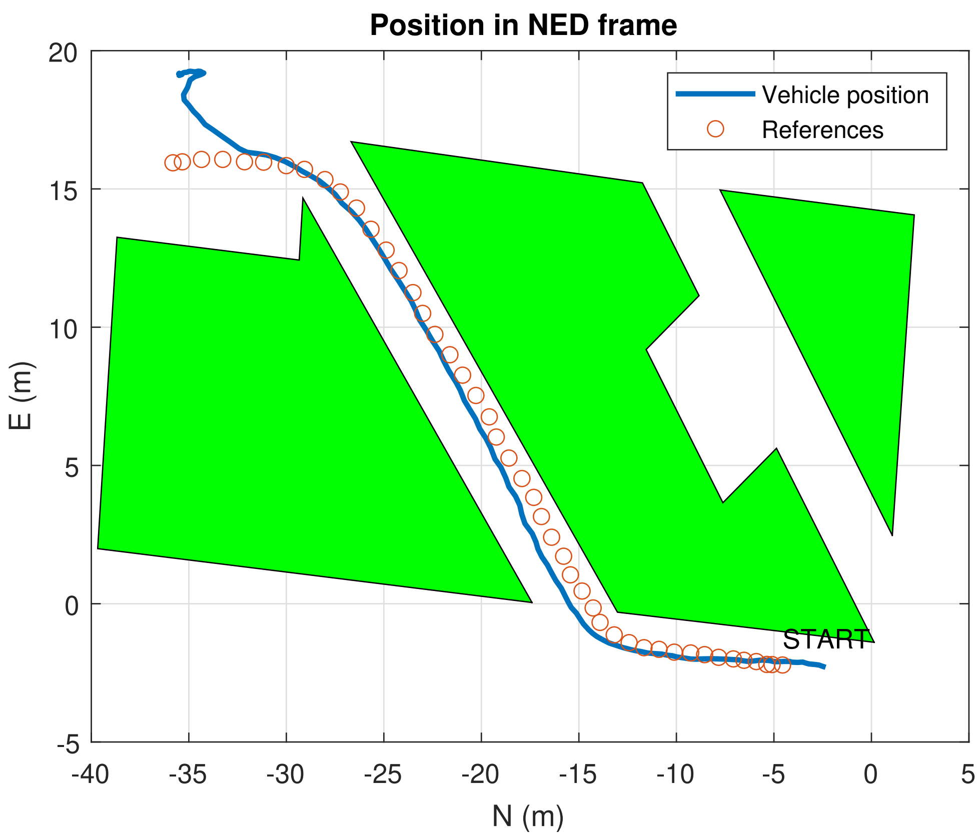


Figure 7.19: Vehicle positions and reference signals in NE plane

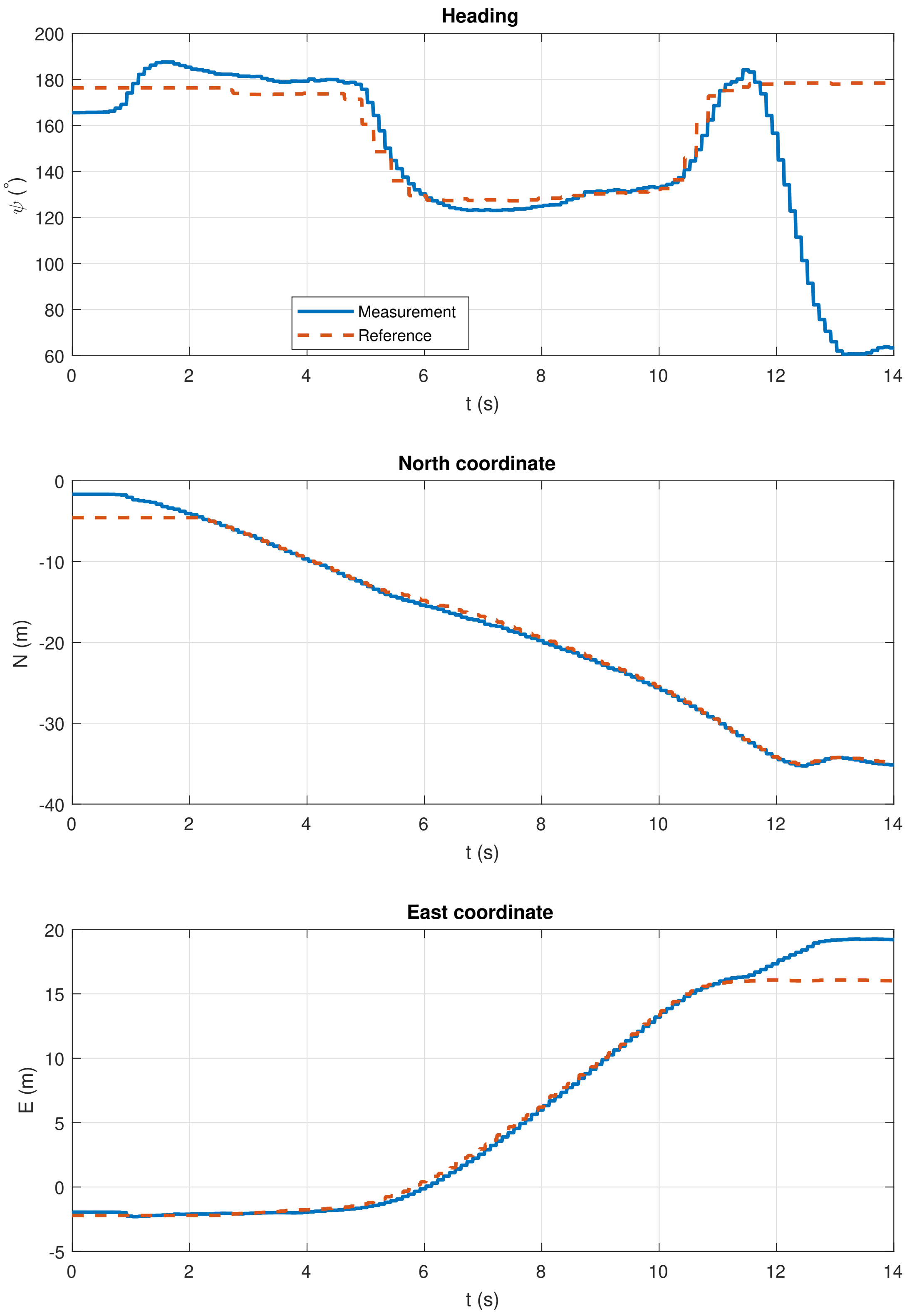


Figure 7.20: Measurement and reference signals comparison

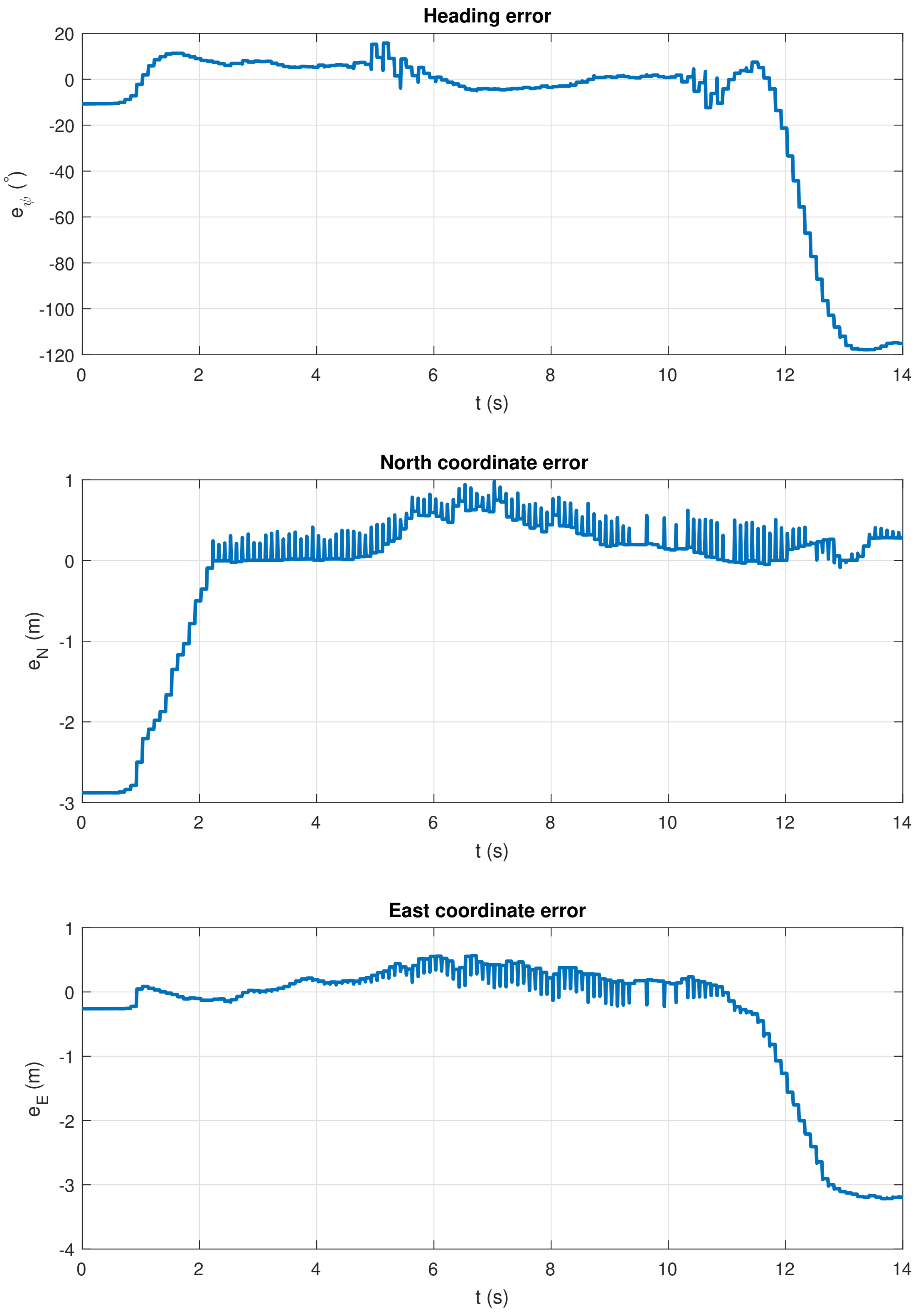
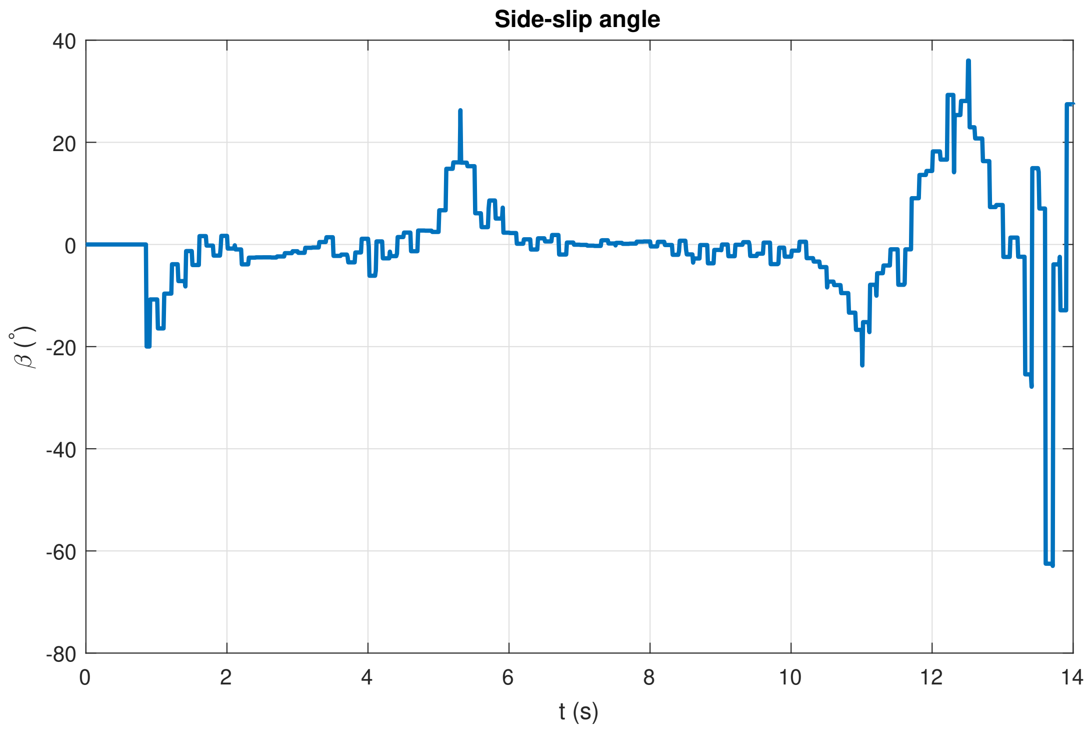


Figure 7.21: Control errors



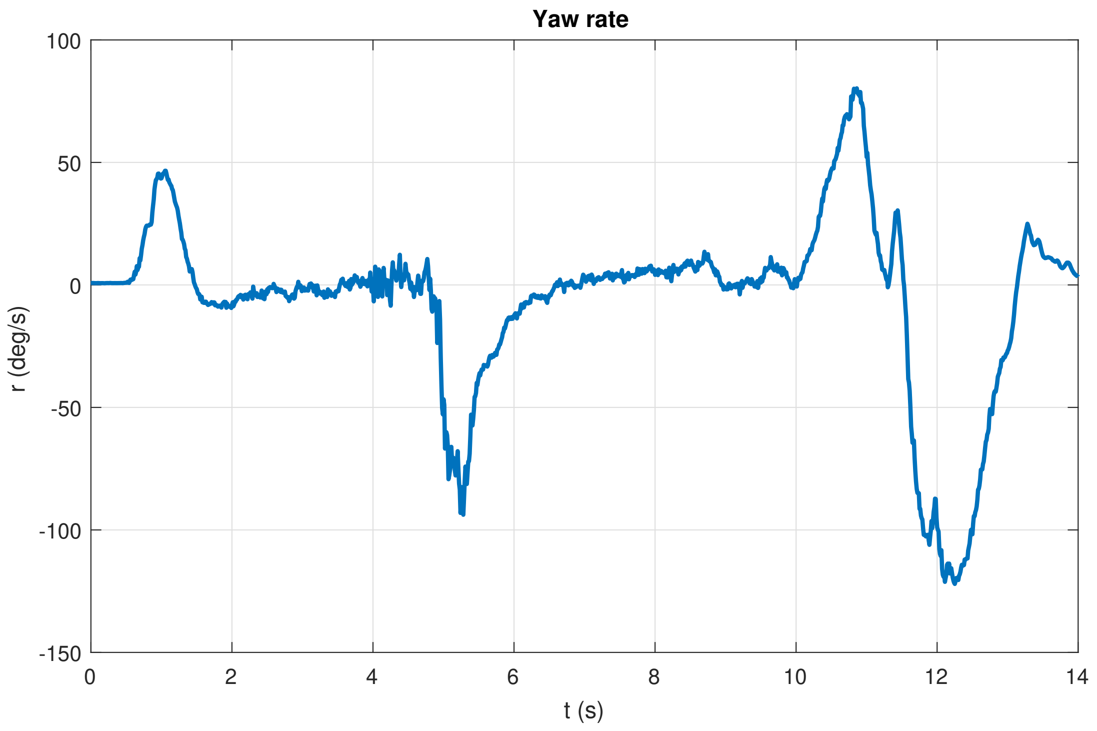


Figure 7.22: Untracked states of the system

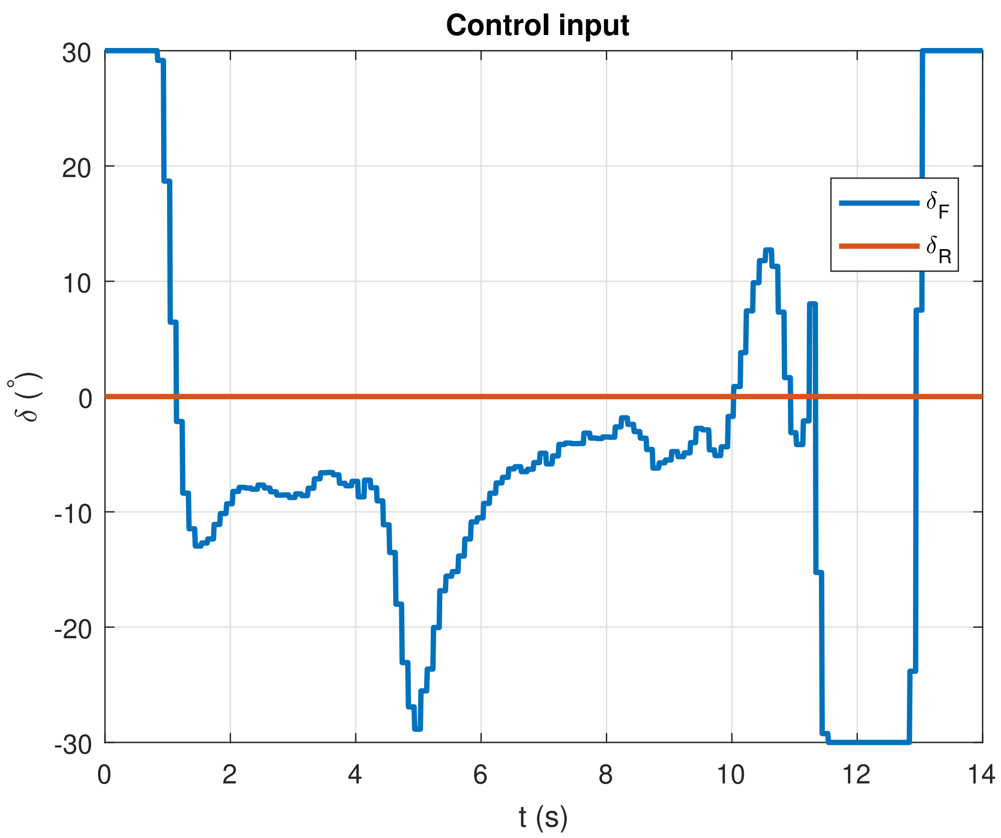


Figure 7.23: Control action of the algorithm

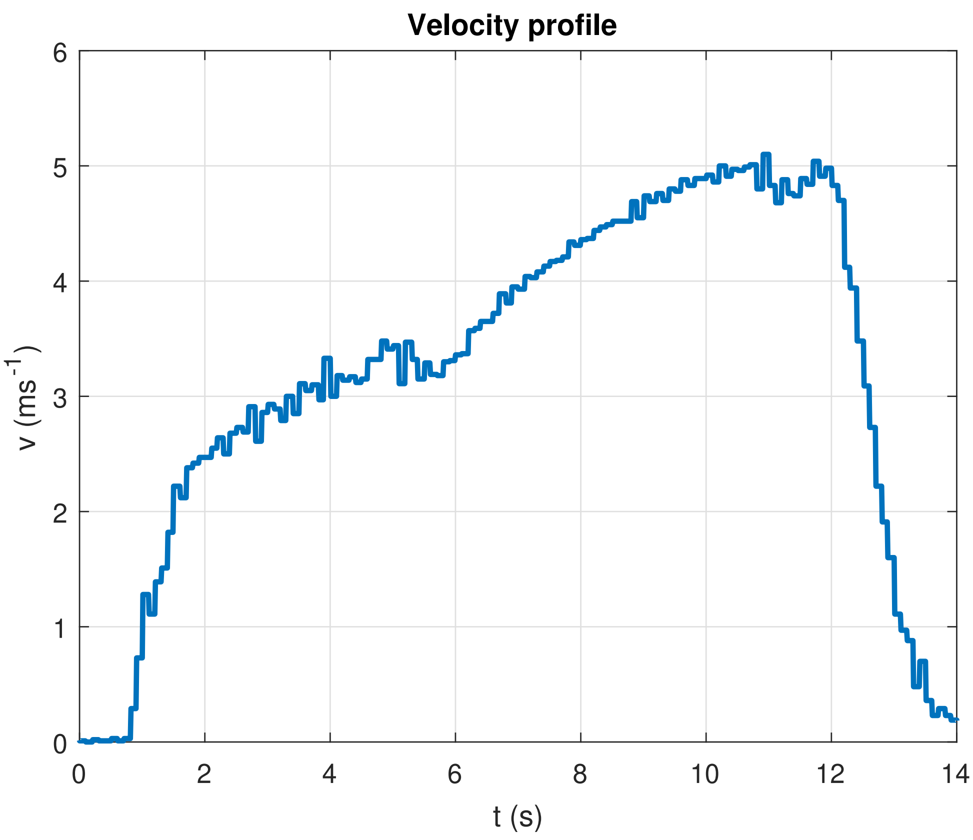


Figure 7.24: Velocity profile

7.3.3 Comparison

Figure 7.25 shows that the MPC with both axle steering enabled performs the best in the test scenario. The MPC algorithms overall had better performance in this test.
The Stanley control law inspired algorithms used in this test can't handle the nonideal steering of the vehicle and have a steady deviation from the path.
The MPC algorithms have got a known issue. The heading reference signal is wrapped to region $[-180°; 180°)$ and the MPC algorithms try to turn the vehicle about a full circle to match the reference when the reference signal overflows the range $[-180°; 180°)$.

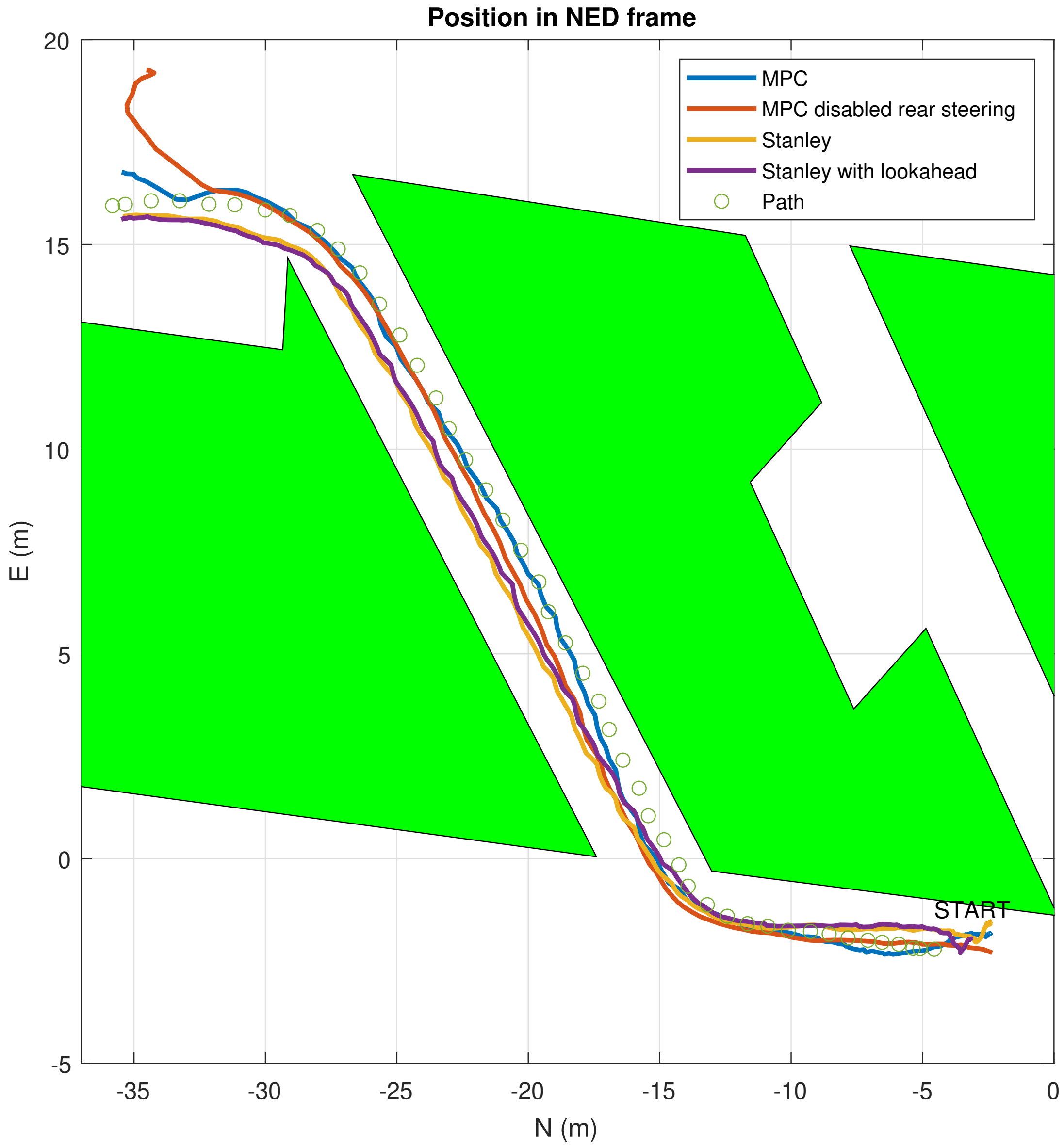


Figure 7.25: Comparison of deployed algorithms

Chapter 8

Conclusion

8.1 Discussion

The purpose of this work was to design a suitable control algorithm for over-actuated vehicle platform that would track the reference path and then compare it to a baseline algorithm.The baseline algorithm used in this work was the modified Stanley Control Law [HTMT07] from Stanford Unirversity. The designed control algorithm was the MPC based algorithm. This algorithm was capable of exploiting the features of the over-actuated platform and also has the effect of the anticipation. That means that it can react to curves on the path before the curve starts. This resulted in better tracking performance then algorithm without preview.
The algorithms were also succesfully deployed on the vehicle platform, which could also serve as the demonstrator for these algorithms.
The comaprison of the MPC based algorithm and baseline algorithm shows that the anticipation capability of MPC results in tighter tracking of the reference path and also the MPC could be suitable algorithm to use with the over-actuated vehicles. The MPC based algorithm also has capability to add constraint to the vehicle motion so that the path tracking much likely doesn't end up in dangerous situations.

8.2 Future Work

- **Adaptive MPC algorithm**
  There is a possibility to enahnce basic MPC algorithm with external setting of cornering stiffnesses from the estimator described for example in this article [VCH+21]
- **Nonlinear MPC**
  The MPC can be also formulated using nonlinear predictor and with that also nonlinear QP solver
- **Adding torque vectoring** There is also a possibility to include information about velocity and torque inside the MPC framework and further enahance it to include torque vectoring to the optimization problem. This possibility would also require more suitable vehicle platform.
- **Soft constraints on the vehicle position** The MPC framework can be also enhanced with sof constraints which could reflect for example dimensions of the lane on the road or other vehicles in the traffic.
- **Deploying of the algorithm in the real vehicle** Lathough, the algorithm was deployed oon the real vehicle platform it could be interesting to deploy it in the real commercially available vehicle with some modifications.

Appendix A

Bibliography

[BFP$^+$20] Michal Bahnik, Dominik Filyo, David Pekarek, Martin Vlasimsky, Jan Cech, Tomas Hanis, and Martin Hromcik, *Visually assisted anti-lock braking system*, 2020 IEEE Intelligent Vehicles Symposium (IV), 2020, pp. 1219–1225.

[Boh22] Marek Boháč, *Návrh systému pro plánování mise autonomního vozidla* (en), Accepted: 2022-06-09T22:57:29Z.

[BSA$^+$17] Sofiane Bacha, Ramzi Saadi, Mohamed Yacine Ayad, Abdennacer Aboubou, and Mebarek Bahri, *A review on vehicle modeling and control technics used for autonomous vehicle path following*, Mar 2017.

[BTKS] M. Bohac, T. Twardzik, A. Konopisky, and J. Svancar, *Hw architecture of tomi2*, https://gitlab.fel.cvut.cz/hanistom/toyota-cross-modal-training/-/wikis/HW-overview, Accesed: 01-01-2023.

[CHK$^+$21] Jan Cech, Tomas Hanis, Adam Kononisky, Tomas Rurtle, Jan Svancar, and Tomas Twardzik, *Self-supervised learning of camera-based drivable surface roughness*, 2021 IEEE Intelligent Vehicles Symposium (IV), 2021, pp. 1319–1325.

[CW14] Yan Chen and Junmin Wang, *Design and experimental evaluations on energy efficient control allocation methods for overactuated electric vehicles: Longitudinal motion case*, IEEE/ASME Transactions on Mechatronics **19** (2014), no. 2, 538–548.

[EKHH20] Denis Efremov, Martin Klaučo, Tomáš Haniš, and Martin Hromčík, *Driving envelope definition and envelope protection us-*

ing model predictive control, 2020 American Control Conference (ACC), 2020, pp. 4875–4880.

[EZKH21] Denis Efremov, Yehor Zhyliaiev, Bohdan Kashel, and Tomáš Haniš, *Lateral driving envelope protection using cascade control*, 2021 21st International Conference on Control, Automation and Systems (ICCAS), 2021, pp. 1440–1446.

[HRK10] J. Hrbáček, T. Ripel, and Jiri Krejsa, *Ackermann mobile robot chassis with independent rear wheel drives*, Oct 2010.

[HTMT07] Gabriel M. Hoffmann, Claire J. Tomlin, Michael Montemerlo, and Sebastian Thrun, *Autonomous automobile trajectory tracking for off-road driving: Controller design, experimental validation and racing*, 2007 American Control Conference, 2007, pp. 2296–2301.

[KFT$^+$08] Yoshiaki Kuwata, Gaston A. Fiore, Justin Teo, Emilio Frazzoli, and Jonathan P. How, *Motion planning for urban driving using rrt*, 2008 IEEE/RSJ International Conference on Intelligent Robots and Systems, Sep 2008, p. 1681–1686.

[Kon22] Adam Konopiský, *Predikce vlastností povrchu z obrazu s využitím samoučení* (en), Accepted: 2022-06-09T22:56:07Z.

[LJKK19] Kibeom Lee, Seungmin Jeon, Heegwon Kim, and Dongsuk Kum, *Optimal path tracking control of autonomous vehicle: Adaptive full-state linear quadratic gaussian (lqg) control*, IEEE Access **7** (2019), 109120–109133.

[LLVT17] Chang Liu, Seungho Lee, Scott Varnhagen, and H. Eric Tseng, *Path planning for autonomous vehicles using model predictive control*, 2017 IEEE Intelligent Vehicles Symposium (IV), Jun 2017, p. 174–179.

[LLZ$^+$19] Shaosong Li, Zheng Li, Bangcheng Zhang, Shunhang Zheng, Xiaohui Lu, and Zhixin Yu, *Path tracking for autonomous vehicles based on nonlinear model: Predictive control method*, Apr 2019, pp. 2019–01–1017.

[Mata] MathWorks, *Embedded coder*, `https://www.mathworks.com/help/optim/ug/lsqnonlin.html`, Accesed: 03-01-2023.

[Matb] ———, *Embedded coder*, `https://www.mathworks.com/products/embedded-coder.html`, Accesed: 03-01-2023.

[Matc] ———, *quadprog*, `https://www.mathworks.com/help/optim/ug/quadprog.html`, Accesed: 03-01-2023.

[MB16] Rashid Mehmood and Umar Bhatti, *Coarse determination of attitude using a single gps receiver*, Ph.D. thesis, Apr 2016.

[PAT] *Path message definition*, `https://docs.ros2.org/foxy/api/nav_msgs/msg/Path.html`, Accesed: 03-01-2023.

[PN14] Plamen Petrov and Fawzi Nashashibi, *Modeling and nonlinear adaptive control for autonomous vehicle overtaking*, IEEE Transactions on Intelligent Transportation Systems **15** (2014), no. 4, 1643–1656.

[PSLM17] Gonçalo Collares Pereira, Lars Svensson, Pedro F. Lima, and Jonas Mårtensson, *Lateral model predictive control for over-actuated autonomous vehicle*, 2017 IEEE Intelligent Vehicles Symposium (IV), Jun 2017, pp. 310–316.

[RCCT$^+$13] Luis I. Reyes Castro, Pratik Chaudhari, Jana Tůmová, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Incremental sampling-based algorithm for minimum-violation motion planning*, 52nd IEEE Conference on Decision and Control, Dec 2013, p. 3217–3224.

[RH05] A. Richards and J. How, *Mixed-integer programming for control*, Proceedings of the 2005, American Control Conference, 2005., Jun 2005, p. 2676–2683 vol. 4.

[SNK19] Saeid Sedighi, Duong-Van Nguyen, and Klaus-Dieter Kuhnert, *Guided hybrid a-star path planning algorithm for valet parking applications*, 2019 5th International Conference on Control, Automation and Robotics (ICCAR), Apr 2019, p. 570–575.

[SZC$^+$20] Yunxiao Shan, Boli Zheng, Longsheng Chen, Long Chen, and De Chen, *A reinforcement learning-based adaptive path tracking approach for autonomous driving*, IEEE Transactions on Vehicular Technology **69** (2020), no. 10, 10581–10595.

[TRCK$^+$13] Jana Tůmová, Luis I. Reyes Castro, Sertac Karaman, Emilio Frazzoli, and Daniela Rus, *Minimum-violation ltl planning with conflicting specifications*, 2013 American Control Conference, Jun 2013, p. 200–205.

[TS] Mario Theers and Mankaran Singh, `https://thomasfermi.github.io/Algorithms-for-Automated-Driving/Control/PurePursuit.html`.

[TYKAM15] Seyed Tabatabaei, Aghil Yousefi-Koma, Moosa Ayati, and Seyed Mohtasebi, *Three dimensional fuzzy carrot-chasing path following algorithm for fixed-wing vehicles*, Oct 2015, pp. 784–788.

[TYZ$^+$20] Luqi Tang, Fuwu Yan, Bin Zou, Kewei Wang, and Chen Lv, *An improved kinematic model predictive control for high-speed path tracking of autonomous vehicles*, IEEE Access **8** (2020), 51400–51413.

[VCH+21] David Vosahlik, Jan Cech, Tomas Hanis, Adam Konopisky, Tomas Rurtle, Jan Svancar, and Tomas Twardzik, *Self-supervised learning of camera-based drivable surface friction*, 2021 IEEE International Intelligent Transportation Systems Conference (ITSC), 2021, pp. 2773–2780.

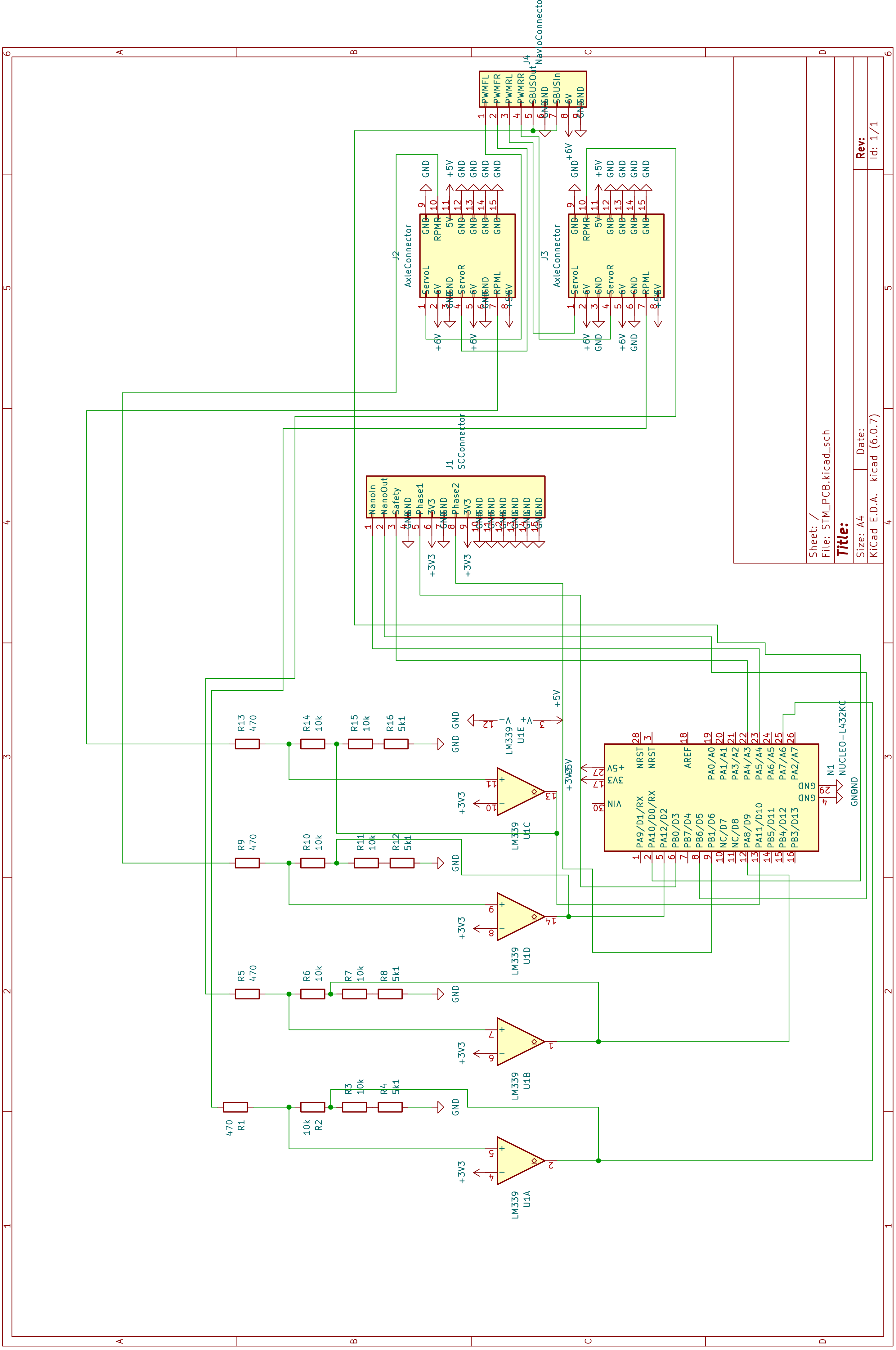
[YGP+19] Jiaxing Yu, Xuexun Guo, Xiaofei Pei, Zhenfu Chen, Maolin Zhu, and Bian Gong, *Robust model predictive control for path tracking of autonomous vehicle*, Apr 2019, pp. 2019–01–0693.

[Şe17] Murat Şenipek, *Development of an object-oriented design, analysis and simulation software for a generic air vehicle*, Ph.D. thesis, Sep 2017.

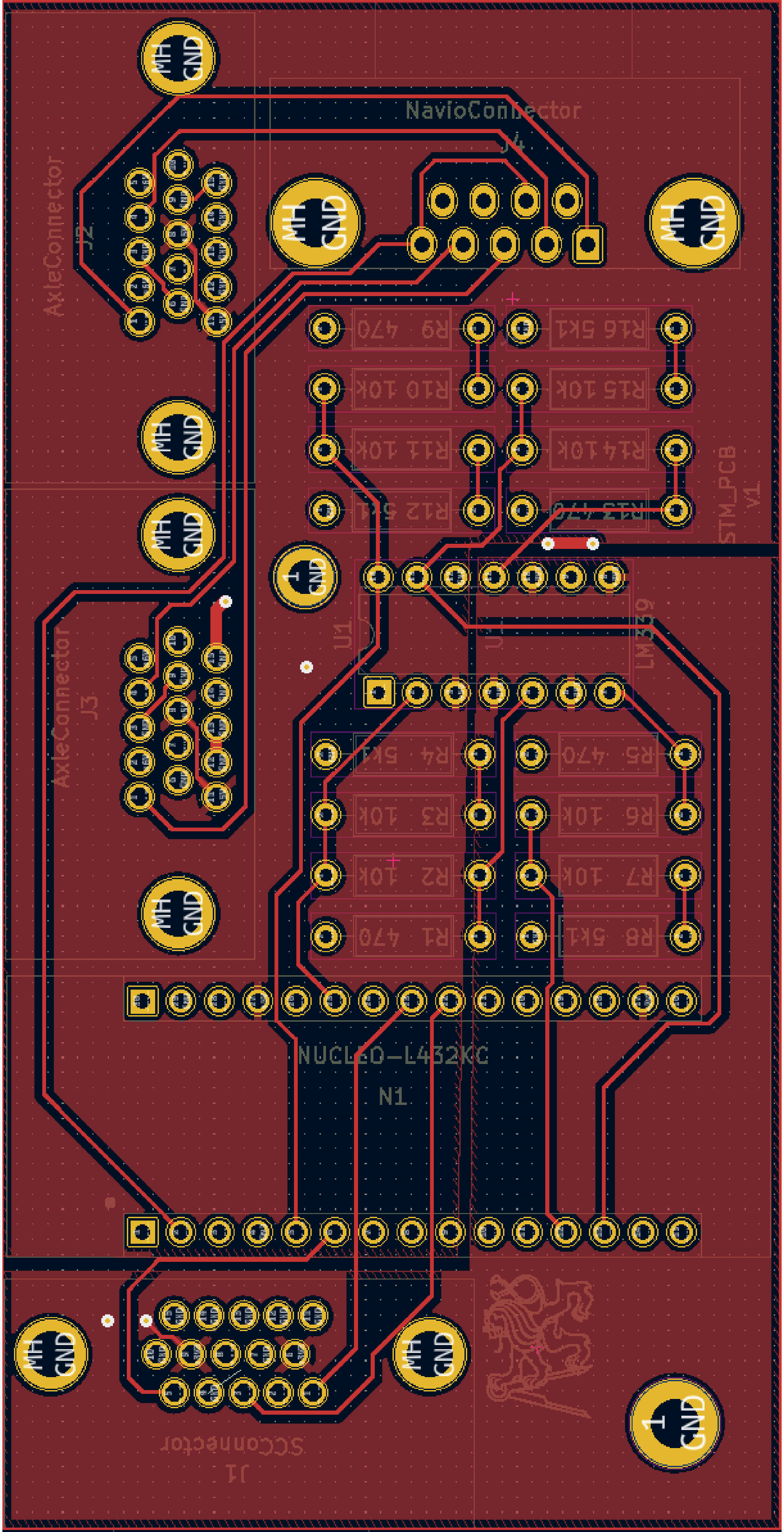
Appendix B

PCB Designs

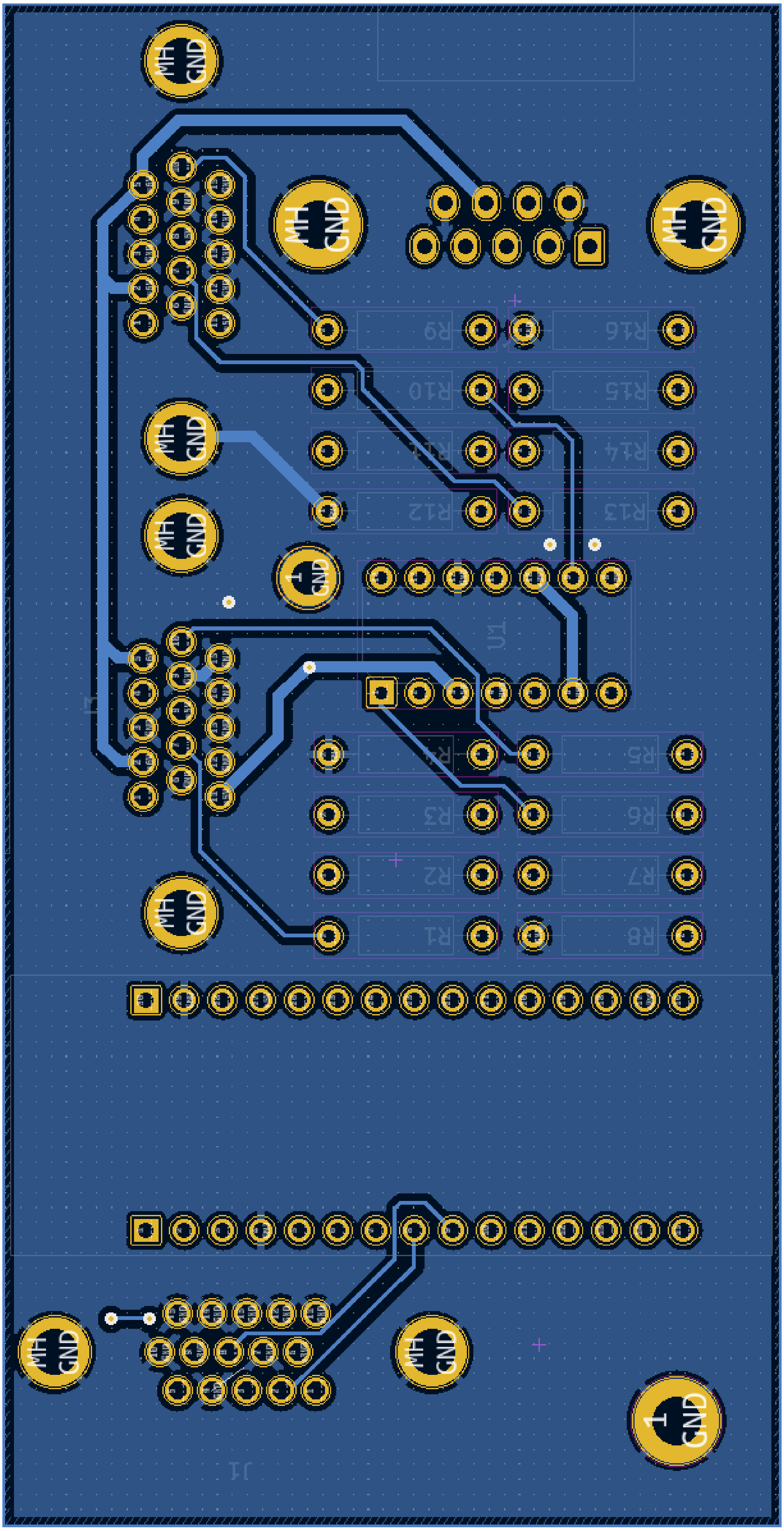
B.1 STM Nucleo breakout board



J4
NavioConnector
PWMFL
PWMFR
PWMRL
PWMRR
SBUSOut
GND
SBUSIn
6V
GND
J2
AxleConnector
J3
AxleConnector
ServoL
6V
GND
ServoR
RPML
5V
RPMR
+6V
+5V
GND
J1
SCConnector
NanoIn
NanoOut
Safety
Phase1
3V3
Phase2
+3V3
N1
NUCLEO-L432KC
NRST
AREF
PA0/A0
PA1/A1
PA3/A2
PA4/A3
PA5/A4
PA6/A5
PA7/A6
PA2/A7
PA9/D1/RX
PA10/D0/RX
PA12/D2
PB0/D3
PB7/D4
PB6/D5
PB1/D6
NC/D7
NC/D8
PA8/D9
PA11/D10
PB5/D11
PB4/D12
PB3/D13
VIN
+5V
3V3
LM339
U1A
U1B
U1C
U1D
U1E
R1 470
R2 10k
R3 10k
R4 5k1
R5 470
R6 10k
R7 10k
R8 5k1
R9 470
R10 10k
R11 10k
R12 5k1
R13 470
R14 10k
R15 10k
R16 5k1
Sheet: /
File: STM_PCB.kicad_sch
Title:
Size: A4
Date:
Rev:
KiCad E.D.A. kicad (6.0.7)
Id: 1/1

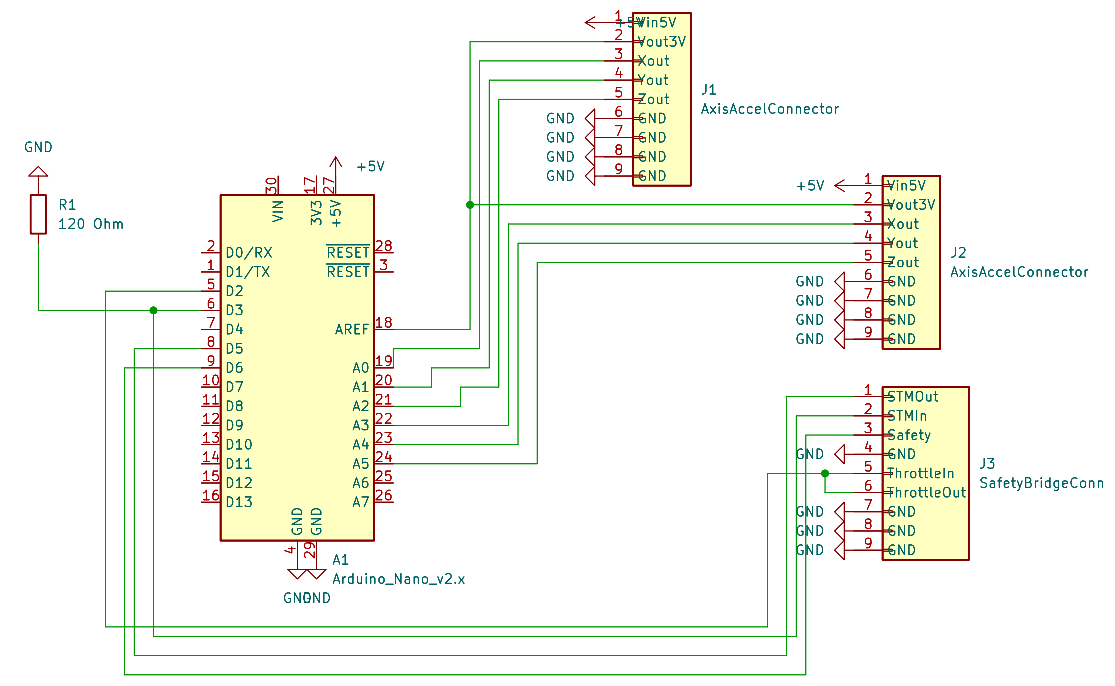


NavioConnector
J4
AxleConnector
J2
AxleConnector
J3
R9 470
R10 10k
R11 10k
R12 5k1
R16 5k1
R15 10k
R14 10k
R13 470
U1
LM339
STM_PCB v1
R4 5k1
R3 10k
R2 10k
R1 470
R5 470
R6 10k
R7 10k
R8 5k1
NUCLEO-L432KC
N1
SCConnector
J1
MH GND
1 GND



MH GND
R1
R2
R3
R4
R5
R6
R7
R8
R9
R10
R11
R12
R13
R14
R15
R16
U1
J1

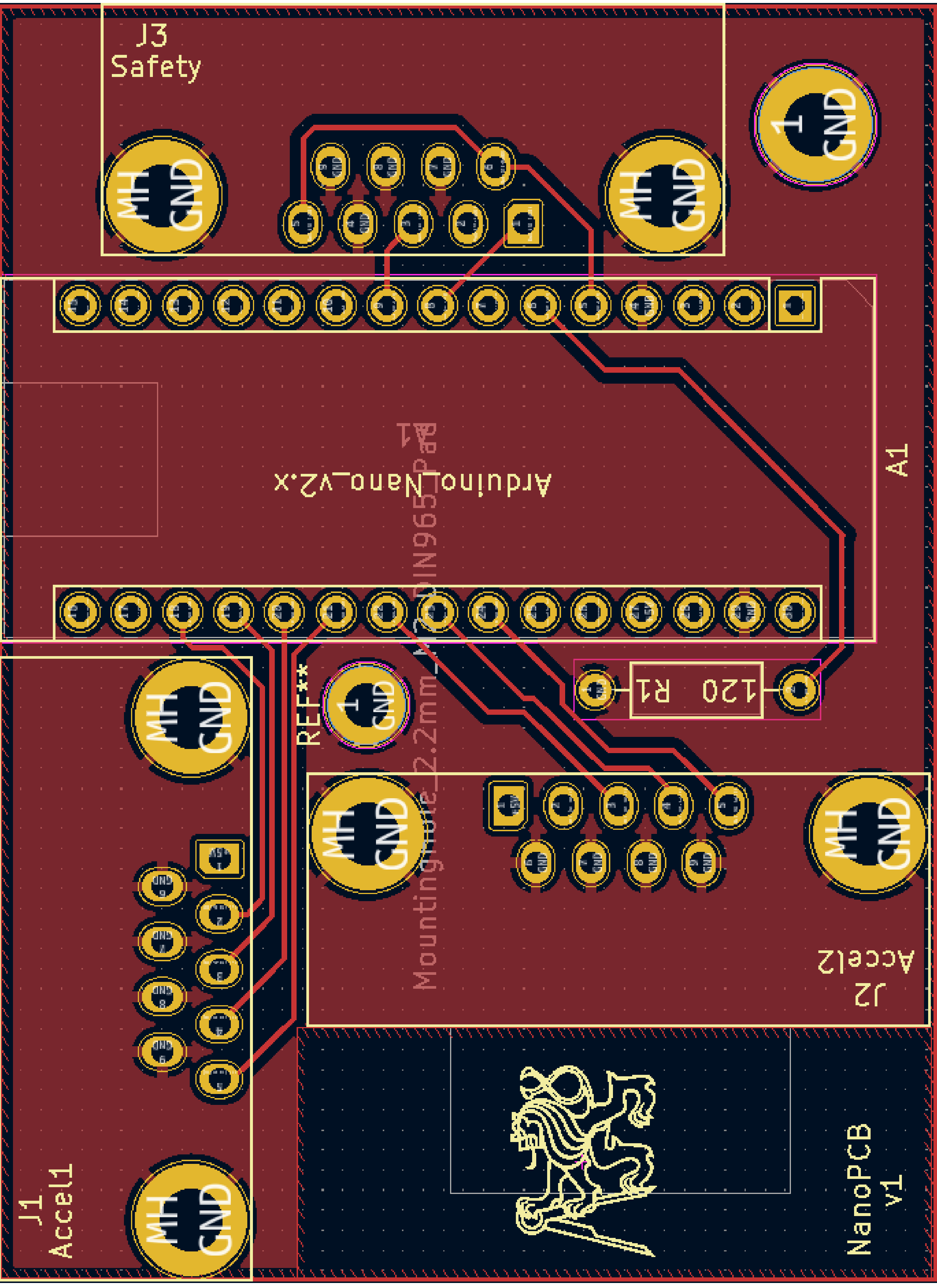
B.2 Arduino Nano breakout board

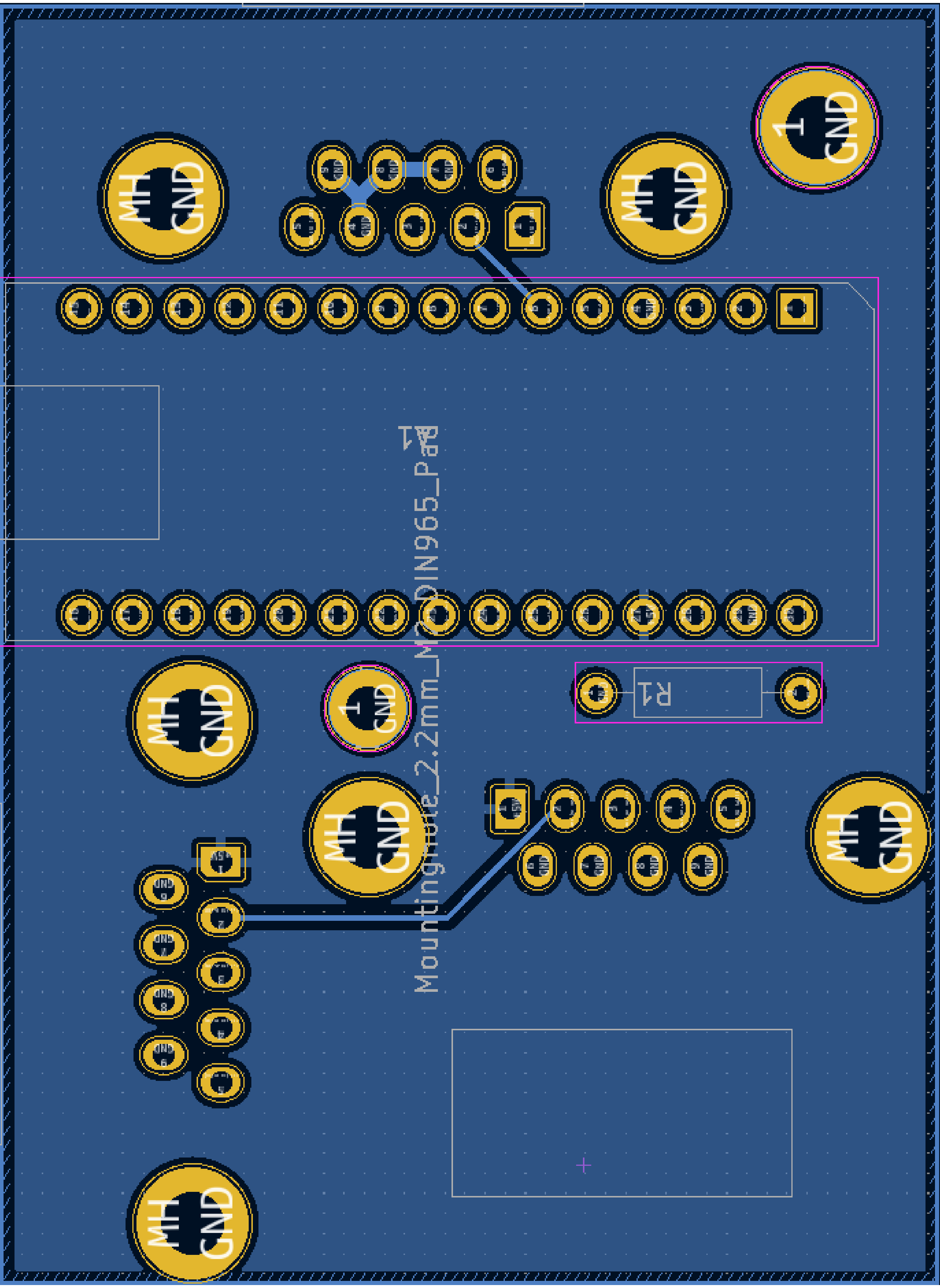


Sheet: /
File: Nano_PCB.kicad_sch

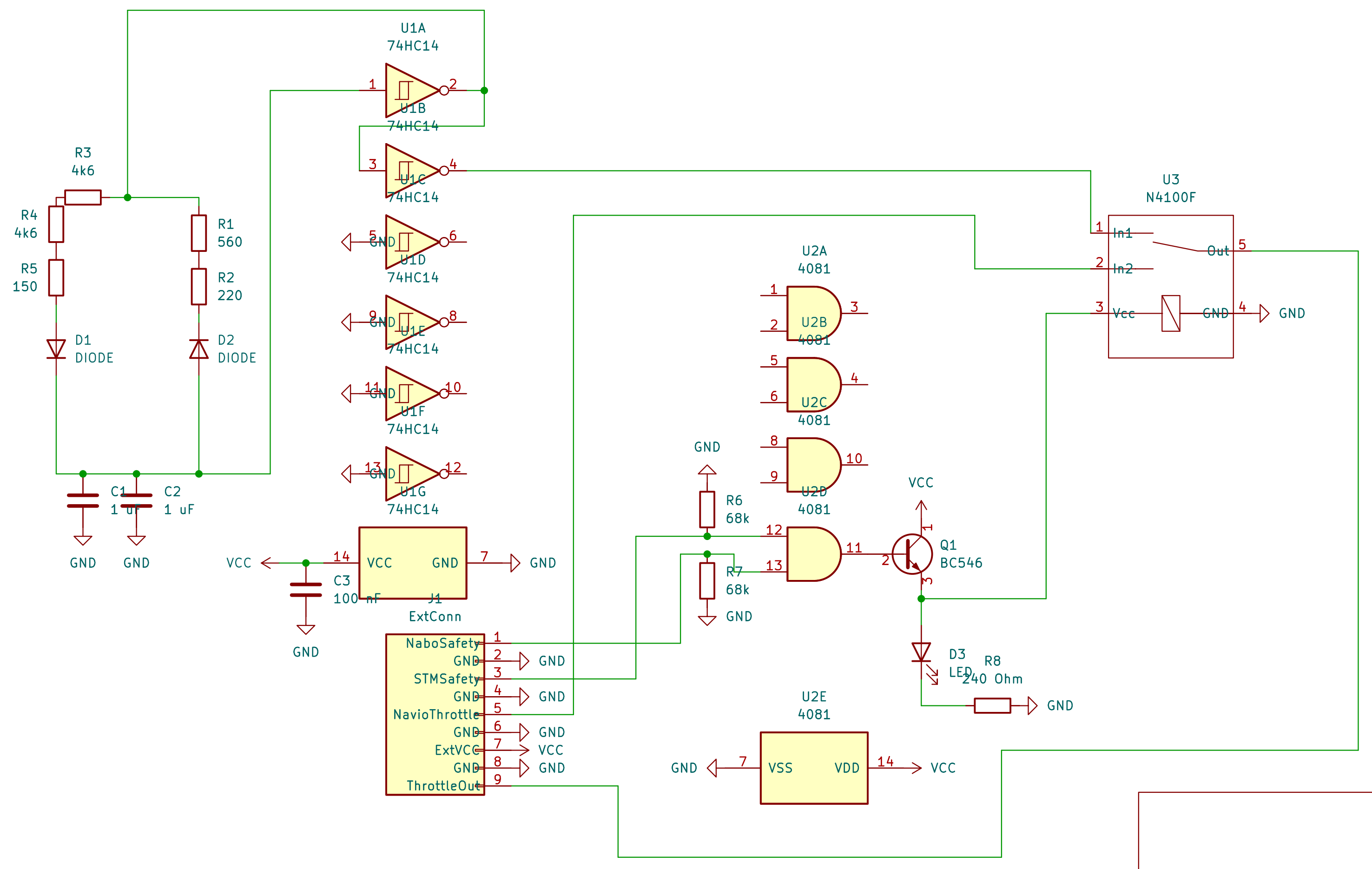
Title:

Size: A4	Date:	**Rev:**
KiCad E.D.A. kicad (6.0.7)		Id: 1/1


J3
Safety
MH GND
GND 1
A1
Arduino_Nano_v2.x
REF**
120 R1
J2
Accel2
J1
Accel1
NanoPCB
v1
MountingHole_2.2mm_M2_DIN965_Pad


MH GND
1 GND
T1
Pad
MountingHole_2.2mm_M2_DIN965_Pad
R1

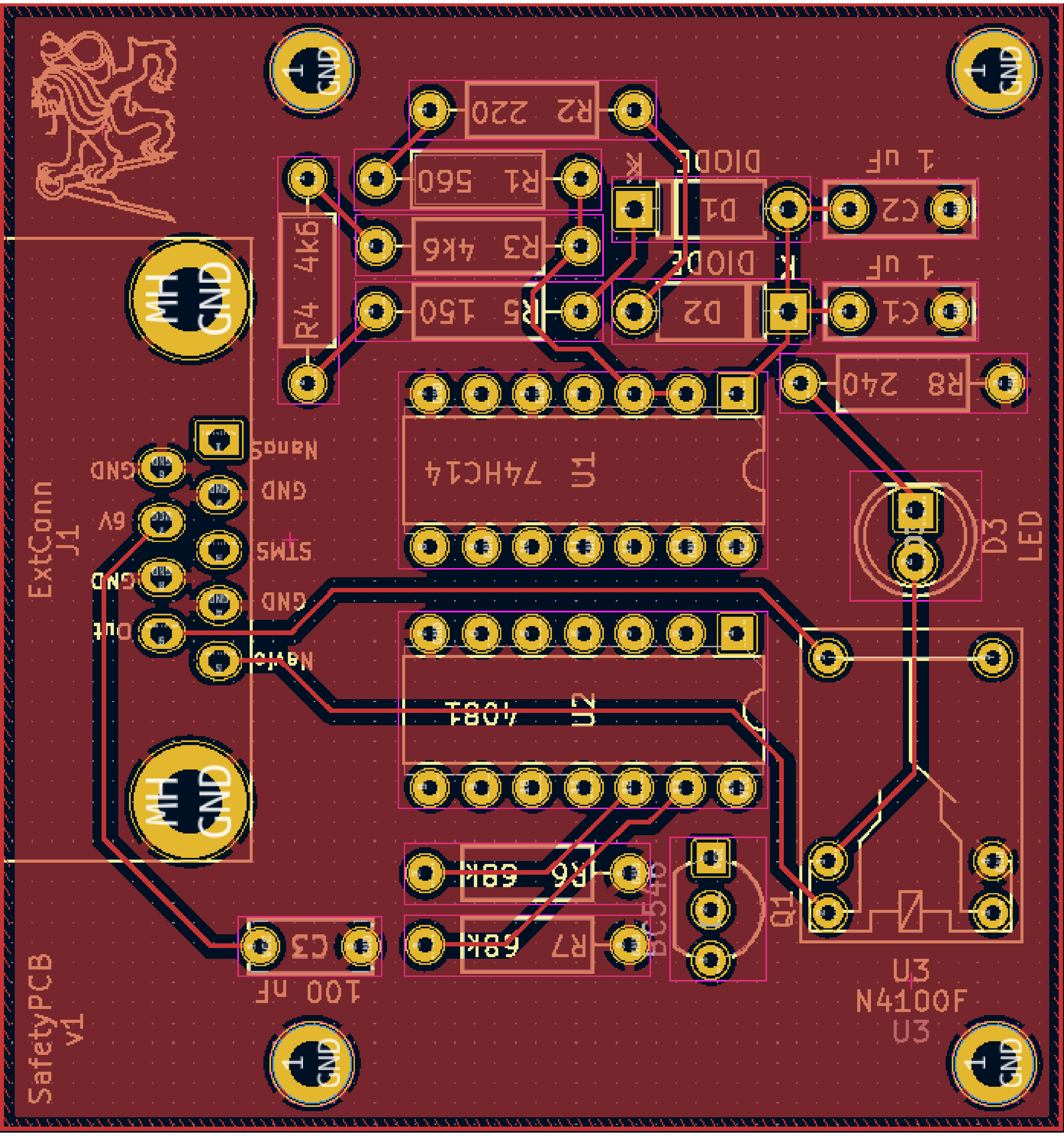
B.3 Safety circuit board

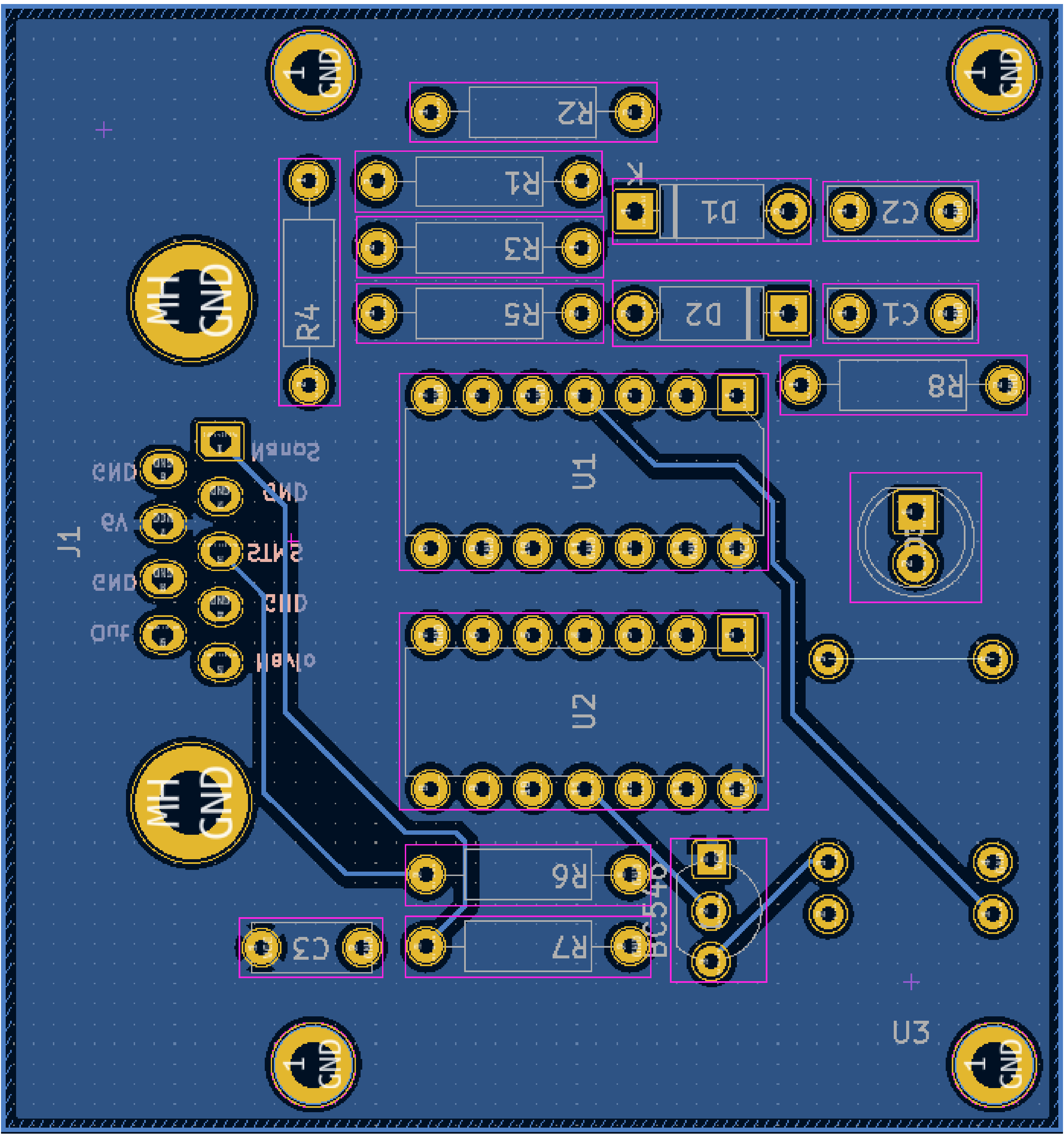


Sheet: /
File: Safety_PCB.kicad_sch

Title:

Size: A4 Date: **Rev:**
KiCad E.D.A. kicad (6.0.7) Id: 1/1


R2 220
R1 560
R3 4k6
R4 4k6
R5 150
D1 DIODE
D2 DIODE
C2 1 uF
C1 1 uF
R8 240
U1 74HC14
U2 4081
D3 LED
ExtConn J1
GND
6V
SW1S
Out
Q1 BC546
R6 68k
R7 68k
C3 100 nF
U3 N4100F
SafetyPCB v1
MH GND


GND
MH GND
R2
R1
R3
R5
R4
K
D1
D2
C2
C1
R8
U1
U2
J1
GND
9V
GND
Out
GND
GND
R6
R7
C3
BC546
U3

MASTER‘S THESIS ASSIGNMENT

I. Personal and study details

Student's name:	**Švancar Jan**	Personal ID number:	**474467**
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:

Autonomous vehicle trajectory tracking algorithms

Master’s thesis title in Czech:

Algoritmy vedení po trati pro autonomní vozidlo

Guidelines:

The objective of this thesis is to review and augment existing path/trajectory tracking algorithm for autonomous vehicles. The developed algorithms will be customized for over-actuated sub-scale demonstration platform with four-wheel steering capabilities. The resulting algorithm will be deployed on embedded hardware to directly control the sub-scale platform.
The thesis will be consisting of following points:
1) Get familiar with trajectory and path tracking algorithms.
2) Select and implement baseline trajectory tracking algorithm like Stanley algorithm.
3) Develop trajectory tracking algorithm reflecting over-actuated vehicle platform.
4) Verification and comparison of developed algorithms with respect to baseline implementation.

Bibliography / sources:

[1] G. M. Hoffmann, C. J. Tomlin, M. Montemerlo, and S. Thrun. Autonomous Automobile Trajectory Tracking for Off-Road Driving: Controller Design, Experimental Validation and Racing. In 2007 American Control Conference, pages 2296–2301, July 2007.
[2] Jan Filip, Trajectory Tracking for Autonomous Vehicles
[3] P. Falcone, F. Borrelli, J. Asgari, H. E. Tseng, and D. Hrovat. Predictive Active Steering Control for Autonomous Vehicle Systems. IEEE Transactions on Control Systems Technology, 15(3):566–580, May 2007.
[4] Robert Bosch GmbH - Bosch automotive handbook - Plochingen, Germany : Robet Bosch GmbH ; Cambridge, Mass. : Bentley Publishers
[5] Dieter Schramm, Manfred Hiller, Roberto Bardini – Vehicle Dynamics – Duisburg 2014

Name and workplace of master’s thesis supervisor:

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

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

Date of master’s thesis assignment: **28.01.2022** Deadline for master's thesis submission: **10.01.2023**

Assignment valid until:
by the end of winter semester 2023/2024

doc. Ing. Tomáš Haniš, Ph.D.	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