NTNU
Kunnskap for en bedre verden

Banefølging for selvgående terrengkjøretøy med Model Predicitive Control

Stian Dyrnes

Master i kybernetikk og robotikk
Innlevert: september 2018
Hovedveileder: Kristin Ytterstad Pettersen, ITK

Norges teknisk-naturvitenskapelige universitet
Institutt for teknisk kybernetikk

Abstract

This thesis is about path following for an Unmanned Ground Vehicle (UGV) for the use in an off-road environment. This thesis focuses on the Model Predicitve Control (MPC) approach to path following, the benefits this may have for the control system and the challenges one might face with this approach.

A literature review was conducted for the basic theory of MPC, as well as existing methods for path following problems. Existing methods for the use of MPC in path following problems was also reviewed.

The vehicle model is based on earlier work where the vehicle was modelled for 6-Degrees of Freedom (DOF). Measurements of the vehicle dimensions was made, in order to make a more accurate estimation of the moment of inertia. A step response experiment was performed on the UGV in order to determine the model parameters for this dynamic vehicle model.

A path definition was introduced to serve the path following objective, and a new optimization formulation was derived as a basis for MPC control laws for path following problems. This optimization formulation was used to derive a variety of control laws with varying degree of accuracy and computational complexity. Further, algorithms realizing the MPC control laws was derived, and the solvers used to solve the optimization was presented.

The functionality of the optimization formulation was tested, and simulation results of the UGV using the MPC control laws, both with and without disturbances, were presented. The computation time of the control laws with different solvers was recorded, and compared with each other.

The performance and run time of the controllers was compared, and the challenges of implementing these control laws on the UGV was discussed. These challenges were also compared to the existing path following controllers.

It was concluded that the optimization formulation derived in this thesis, worked as intended for path following MPCs. Further it was concluded that the both the 3-DOF and the simplified 6-DOF MPC control laws showed performed well in simulations both with and without disturbances, however the run time with the current solvers was too large to be implemented on the UGV.

Preface

This thesis is written as a final master thesis in the two year master's program within Cybernetics and Robotics at Norwegian University of Science and Technology. The task for this thesis was given by the Norwegian Defence Research Establishment in cooperation with Norwegian University of Science and Technology.

The goal of this thesis was to derive model predictive control laws for path following of off-road unmanned ground vehicles. Further the goal was to look in to whether these control laws could be implemented on the Norwegian Defence Research Establishment's unmanned ground vehicle OLAV.

I would like to thank my supervisors Kristin Y. Pettersen from Norwegian University of Science and Technology and Kim Mathiassen from Norwegian Defence Research Establishment for their guidance and all their contribution to my thesis. I would also like to give a special thanks to Giorgio D. Kwame Minde Kufoalor for taking the time to discuss my control control law with me, and helping me by recommending optimization solvers. I would also like to thank everyone else at Norwegian Defence Research Establishment who have contributed to this thesis. At last I would like to thank my parents for supporting me through my studies, and especially to my father for always taking the time to discuss topics I have struggled with.

Table of Contents

Abstract i

Preface ii

Table of Contents iv

List of Tables v

List of Figures ix

Abbreviations xi

1 Introduction 1
- 1.1 Motivation . . . 1
- 1.2 Assumptions . . . 1
- 1.3 Background and contribution . . . 2
- 1.4 Notation . . . 3

2 Literature Review 5
- 2.1 Model Predictive Control . . . 5
  - 2.1.1 Linear Model Predictive Control . . . 5
  - 2.1.2 Nonlinear Model Predictive Control . . . 8
  - 2.1.3 Non-conventional approaches to Model Predictive Control . . . 9
- 2.2 Path following control . . . 10
  - 2.2.1 Kinematic path following controller . . . 10
  - 2.2.2 Kinetic path following control . . . 10
  - 2.2.3 MPC for path following problems . . . 10

3 Basic Theory 13
- 3.1 Vehicle Model . . . 13
- 3.2 Testing the vehicle model . . . 15
- 3.3 Model Parameters . . . 16

3.3.1 Moment of inertia . . . 16
3.3.2 Damping and friction coefficients . . . 18

4 MPC **23**
4.1 Reference path . . . 23
4.2 Optimization formulation . . . 24
4.3 Linearized MPC . . . 33
4.3.1 Controllability . . . 33
4.3.2 Control law . . . 36
4.4 Nonlinear MPC for 3-DOF . . . 39
4.5 Simplified Nonlinear MPC for 6-DOF . . . 40
4.6 Multiplexed Nonlinear MPC for 3-DOF . . . 42

5 Implementation **45**
5.1 Algorithms . . . 45
5.1.1 Linearizing MPC . . . 45
5.1.2 Nonlinear MPC for 3-DOF . . . 45
5.1.3 Nonlinear MPC for 6-DOF . . . 46
5.1.4 Nonlinear MMPC for 3-DOF . . . 46
5.2 Solvers and hardware . . . 47

6 Results **49**
6.1 Basic optimization . . . 49
6.2 Linearized MPC for 3-DOF . . . 50
6.3 Nonlinear MPC for 3-DOF . . . 55
6.3.1 APMonitor solvers . . . 55
6.3.2 SciPy solver . . . 63
6.4 Simplified 6-DOF controller . . . 68
6.5 Multiplexed Model Predictive Control . . . 73
6.5.1 APMonitor solver . . . 73
6.5.2 SciPy solver . . . 74
6.6 Run time . . . 79

7 Discussion **83**
7.1 Control law performance . . . 83
7.2 Computation time . . . 86

8 Conclusion and future work **89**
8.1 Conclusion . . . 89
8.2 Future work . . . 90

Bibliography **91**

Appendix **93**

List of Tables

1.1 Description of commonly used symbols 4

3.1 Numerical values for partitions 17

6.1 Computation times for APM solvers for 3-DOF NMPC 80
6.2 Computation times for APM solvers for 3-DOF NMPC 80
6.3 Computation times for SciPy SQP solver for 3-DOF NMPC 80
6.4 Run times for SciPy's SQP solver for 3-DOF MMPC 81

List of Figures

1.1 Current control hierarchy for OLAV from Mathiassen et al. (2016) 2

3.1 Comparison of measured position and simulated position with linear damping model from Dyrnes (2018) 15
3.2 Comparison of measured position and simulated position with nonlinear damping model from Dyrnes (2018) 15
3.3 Vehicle partition in xz-plane from Dyrnes (2018) (not to scale) 16
3.4 Step response of the yaw rate 19
3.5 Measured step response compared to an approximated step response . . . 20
3.6 Time constant for the transfer function 21

4.1 Error from the initial position to the first position on the reference path . . 26
4.2 Error from the initial position to the optimal position on the reference path 27
4.3 Illustration of vehicle overturning from Dyrnes (2018) 42

6.1 Simulation of a simple path following example 50
6.2 Simulation of linearized MPC without disturbances and $q_s = 1$ 51
6.3 Simulation of linearized MPC without disturbances and $q_s = 10$ 51
6.4 Simulation of linearized MPC without disturbances and $q_s = 100$ 52
6.5 Simulation of linearized MPC without disturbances and $q_s = 1000$ 52
6.6 MPC velocity input with $q_s = 1000$ 53
6.7 Simulation of linearized MPC without disturbances and $q_s = 100$ 54
6.8 MPC velocity input with $q_s = 100$ 54
6.9 Simulation of the NMPC for 3-DOF with ideal model 55
6.10 Simulation of the NMPC for 3-DOF with ideal model and initial heading away from the path direction 56
6.11 Simulation of the NMPC for 3-DOF with model disturbances and initial heading towards the path direction 57
6.12 Simulation of the NMPC for 3-DOF with model disturbances and initial heading away from the path direction 57
6.13 Comparison of APOPT and IPOPT solver for 3-DOF Nonlinear MPC . . 58

6.14 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 10$ . . . 59
6.15 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances . . . 60
6.16 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances . . . 60
6.17 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances . . . 61
6.18 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and with disturbances . . . 61
6.19 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and with disturbances . . . 62
6.20 Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ . . . 62
6.21 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 10$ without disturbances . . . 63
6.22 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 20$ without disturbances . . . 64
6.23 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 30$ without disturbances . . . 64
6.24 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 40$ without disturbances . . . 65
6.25 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 10$ with linear path and no disturbances . . . 66
6.26 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 20$ with linear path and no disturbances . . . 66
6.27 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 30$ with linear path and no disturbances . . . 67
6.28 Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 40$ with linear path and no disturbances . . . 67
6.29 Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$ and no disturbances . . . 68
6.30 Comparison of 3-DOF controller and enhanced 3-DOF controller with $\theta = 30°$, starting in $(60,\ 10)$ . . . 69
6.31 Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect on the vehicle . . . 70
6.32 Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect and coefficient errors $\delta \in [1,\ 1.2]$ . . . 71
6.33 Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect and coefficient errors $\delta \in [1,\ 1.2]$ . . . 72
6.34 Simplified 6-DOF MPC with large side slip . . . 73
6.35 Comparison of algorithm 1 and 2 for Multiplexed Model Predictive Control 74
6.36 Multiplexed Model Predictive Control without disturbances with $N = 10$ 75
6.37 Multiplexed Model Predictive Control without disturbances with $N = 20$ 76
6.38 Multiplexed Model Predictive Control without disturbances with $N = 30$ 77

6.39 Multiplexed Model Predictive Control without disturbances with $N = 40$ 78
6.40 Multiplexed Model Predictive Control without disturbances 79

Abbreviations

3D three dimensional. 13, 40

CG Centre of Gravity. 15, 16, 41

DOF Degrees of Freedom. i, 3, 13, 15, 24, 32, 33, 39–42, 45, 46, 68, 72, 79–81, 83, 84, 86, 87, 89

ENU East North Up. 4, 13, 14, 28

FFI Norwegian Defence Research Establishment. 1, 2, 13, 23, 38, 85, 88

GNSS Global Navigation Satellite System. 2

IMU Inertial Measurement Unit. 2

IP Interior Point. 47, 63, 79, 83, 89

ISR Intelligence, Surveillance and Reconnaissance. 1, 85

LOS Line of Sight. 11

LQR Linear Quadratic Regulator. 11

LTI Linear Time Invariant. 6

LTV Linear Time Variant. 5, 11

MHE Moving Horizon Estimator. 9

MMPC Multiplexed Model Predictive Control. 9, 42, 44, 46, 73, 74, 78, 80, 81, 84, 86, 90

MPC Model Predicitve Control. i, 1, 3–11, 13, 23, 25, 28, 33, 37, 38, 40, 42–46, 49, 50, 53, 54, 56, 58, 59, 72, 79, 81, 83–90

NLP Nonlinear Programming. 7, 8, 46, 47

NMPC Nonlinear Model Predictive Control. 8, 10, 11, 39, 40, 45, 46, 55, 79, 80, 83, 86, 89

OLAV Off-road Light Autonomous Vehicle. 1, 2, 13–16, 18, 23, 81, 85–87

PID Proportional Integral Derivative. 83

QP Quadratic Programming. 7, 50, 89

RAM Random Access Memory. 48

ROS Robotic Operating System. 81

SQP Sequential Quadratic Programming. 8, 48, 63, 65, 80, 84, 89, 90

UGV Unmanned Ground Vehicle. i, 1–3, 13, 14, 16, 17, 23, 33, 38–42, 49, 50, 53–55, 69, 84–87, 89, 90

VM Virtual Machine. 48, 81, 87

WCP Wheel Contact Point. 13, 14

Chapter 1

Introduction

1.1 Motivation

Norwegian Defence Research Establishment (FFI) has developed an UGV for the use in Intelligence, Surveillance and Reconnaissance (ISR)-missions. The vehicle has been named Off-road Light Autonomous Vehicle (OLAV). OLAV's missions will mostly be performed in off-road terrain, and the goal is for the vehicle to operate autonomously to reduce personnel risk. In order to achieve this, a path must first be planned, and controllers on the UGV should aim to follow this path.

Because of the harsh conditions in the off-road terrain the UGV is required to traverse in some paths, it is necessary that the vehicle tracks this path as accurately as possible. Even small deviations from the desired path could result in the UGV getting stuck, and could in worst case scenario break the UGV. It should therefore be implemented controllers to accurately track the pre-planned path, and generate the reference velocities and steering angles required to follow the desired path.

OLAV have some functional path following controllers, however there are room for improvement. FFI has desired that more accurate control strategies should be developed and tested in hopes of increasing the accuracy for rough terrain. As a foundation for the new controllers. One idea is to test whether the MPC could help improve the accuracy of the controllers. The goal of this thesis is therefore to derive MPC control laws for the path following objective, and test whether these control laws can be implemented on OLAV.

1.2 Assumptions

The key assumptions for this thesis is that the wheels are in contact with the ground at all times. In reality, small deviations from this may occur, however as these deviations only occur for short time intervals, this assumption holds for most operating conditions of the vehicle.

Another assumption we made in this thesis is that the UGV's low level controllers

are sufficient to achieve the desired velocity and steering angle in one time step, and that no low level dynamics needs to be taken into account for the controller. Tests conducted by FFI shows that the low level controllers are able to achieve the desired velocity and steering angle, however for large changes they are not necessarily reached in one time step. However, as the low level controllers has fast dynamics, this assumption will be used for this thesis, and any deviation from this may be considered as disturbances for the system.

1.3 Background and contribution

This thesis is based on the work and research previously done by FFI on OLAV. The control hierarchy is designed in Mathiassen et al. (2016) and is illustrated in **Fig. 1.1**. This shows the different layers and the information float in the control system.

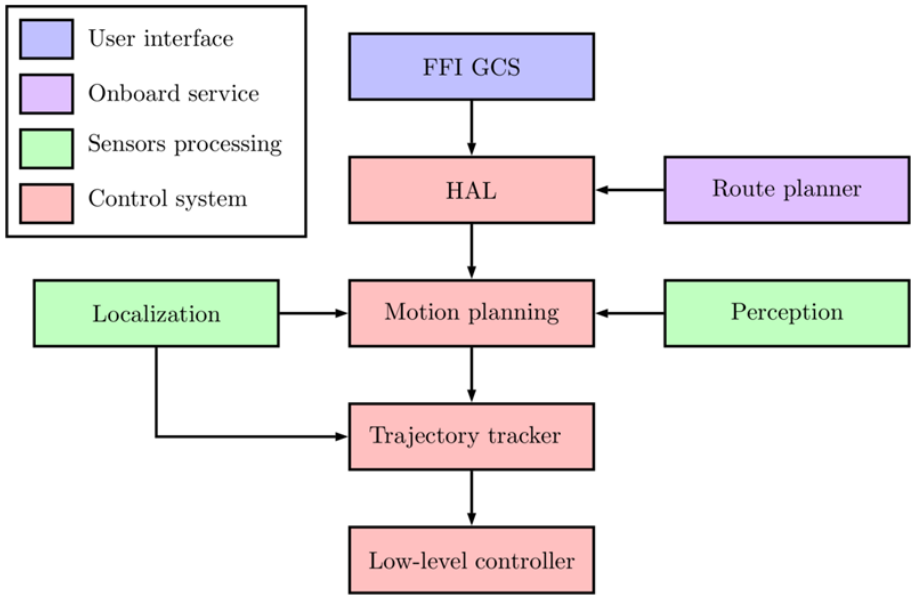

Figure 1.1: Current control hierarchy for OLAV from Mathiassen et al. (2016)

The Trajectory tracker receives the desired path from the motion planning layer, and aims to generate reference inputs which will keep the vehicle on the desired path. These reference signals will be applied to the vehicle by the low level controllers. The low level controllers are working very well, and does not need to be improved. We will therefore not go in dept for these controllers in this thesis.

OLAV is also equipped with sensors such as Global Navigation Satellite System (GNSS) for measurement of position, and Inertial Measurement Unit (IMU) for measuring orientation. These measurements are used in the path following controllers.

The UGV's dynamics was previously modelled for movement in the xz-plane in Auby (2016). Further, the UGV model was expanded in work done by the author in Dyrnes

(2018). The UGV is modelled as a non holonomic control system for 3- and 6-DOF. This means that the movement is constrained by some states, i.e the UGV can not drive directly sideways, and thus, the movement in x and y direction on the ground are constrained by the heading ψ. Further, the vehicle is not controlled for the altitude z, as this will be uniquely given by altitude of the terrain. The data used for model testing was collected with the help of Kim Mathiassen and a script for converting the simulated positions from meters to latitude and longitude coordinates were provided by Magnus Baksaas. Magnus Baksaas also provided a script for converting the measured angles from quaternions to euler angles.

This thesis focuses on the Trajectory tracker layer of **Fig. 1.1**, however some work work will be done for the Motion planning layer. The main contribution of this thesis has been an optimization formulation for path following, as well as several MPC control laws based on this optimization formulation. Further, simulations of the UGV using these control laws have been made in MATLAB and Python, and some solvers have been tested for these simulations. Comparisons of computation time of the control laws with varying solvers have also been compared, and commented upon.

Another contribution of this thesis was more accurate estimations of the model parameters in the UGV model from Dyrnes (2018). This thesis has not made any work in the area of motion planning, however a new path definition was introduced which serves the optimization formulation of the path following.

1.4 Notation

The notation used in this thesis will be similar to that used in Fossen (2011). Bold lower case symbols will be used to describe n-dimensional vectors, as opposed to regular symbols used for variables. E.g. $\boldsymbol{p}$ will be a n-dimensional vector whilst p will be a single variable. Bold upper case symbols will be used to symbolize matrices e.g the rotation matrix $\mathbf{R}$. Some upper case symbols will be used to symbolize constants e.g I for moment of inertia and D for damping coefficient. For some cases where the dimension of matrices needs to be specified, this will be done by the use of subscripts, e.g $\boldsymbol{I}_{2\times 2}$ will denote a two by two identity matrix.

Table 1.1 shows the most commonly used variables for this thesis. Many of these are based on the SNAME notation for marine vessels from 1950 as stated in Fossen (2011), however some are related to the MPC formulation.

Symbol	Description
u	Linear velocity along x-axis in BODY-frame
v	Linear velocity along y-axis in BODY-frame
w	Linear velocity along z-axis in BODY-frame
U	Linear speed $\sqrt{u^2 + v^2 + w^2}$
ϕ	Rotation about x-axis (roll-angle)
θ	Rotation about y-axis (pitch-angle)
ψ	Rotation about z-axis (yaw-angle)
p	Angular velocity about x-axis
q	Angular velocity about y-axis
r	Angular velocity about z-axis
$\boldsymbol{p}$	Position vector $(x \;\; y \;\; z)^T$
$\boldsymbol{\Theta}_{eb}$	Attitude (Euler angles) $(\phi \;\; \theta \;\; \psi)^T$
$\boldsymbol{\nu}^b_{b/e}$	Linear velocity vector $(u \;\; v \;\; w)^T$
$\boldsymbol{\omega}^b_{b/e}$	Angular velocity of BODY-frame in relation to ENU-frame $(p \;\; q \;\; r)^T$
$\mathbf{R}^e_b$	Rotation matrix from ENU to BODY.
T_z	Torque for rotational motion about the z-axis
N	Prediction horizon for the MPC
h	Sampling time

Table 1.1: Description of commonly used symbols

Further, the subscripts of a variable will be used to symbolize for which reference frame it is defined, e.g. x^e will be x given in the East North Up (ENU)-frame. The ENU-frame will for this thesis be a local ENU-frame, i.e the positions are given in meters from a local origo rather than latitude and longitude coordinates. The most common subscripts used in this thesis will be b which represents the reference frame fixed to the vehicle's body (BODY-frame), e which represents ENU-frame and w which represents the WCP-frame. There will also be use of the subscript g which will represent the angle of the ground compared to a flat surface. This is relevant for cases where the ground is not parallel to the ENU-frame, in other words where there is an inclination of the terrain. Another use of subscript will be to symbolize the movement of one coordinate frame in relation to another, e.g. $\boldsymbol{\nu}^b_{b/e}$ will be the velocity of the BODY-frame in relation to the ENU-frame, given in BODY-frame. By default, the orientations $(\phi \;\; \theta \;\; \psi)^T$ will represent the orientation of the BODY compared to ENU, however the subscript w will be used for the orientation of the WCP-frame compared to ENU. E.g ϕ_w is the roll of the WPC-frame.

For cases with large trigonometric functions such as kinematic matrices, the trigonometric functions $\sin$, $\cos$ and $\tan$ will be shortened s, c and t respectively, e.g $\sin\psi$ will be shortened $\mathrm{s}(\psi)$.

Chapter 2

Literature Review

2.1 Model Predictive Control

2.1.1 Linear Model Predictive Control

The theory of MPC is described in Foss and Heirung (2016), as an open loop controller. This is different from traditional controllers in that the control input is not based on feedback from the system states, but rather is found by solving an optimization problem based on measurements of the initial condition $\boldsymbol{x}(0)$. For Linear Time Variant (LTV) systems, the system dynamics is described in discrete time as

$$\boldsymbol{x}(k+1) = A(k)\boldsymbol{x}(k) + B(k)\boldsymbol{u}(k) \tag{2.1}$$

where k is the time step. Further, Foss and Heirung (2016) describes that for time t, the MPC calculates the optimal input sequence for the discrete time steps from $k = 0 \rightarrow N$, where 0 is the current time step and N is the prediction horizon. The prediction horizon is a defined number of time steps the MPC will optimize for, e.g with $N = 10$, the optimization will consider the next ten iterations of the states and inputs. After the optimal input sequence is calculated, the first input of the sequence, i.e $\boldsymbol{u}(0)$ is applied to the system, the system is iterated to time $t + h$, where the states are measured and updated and the optimization is solved for the new time step.

As the linear discrete state space model describes the change of $\boldsymbol{x}$ for any LTV system, Foss and Heirung (2016) use these dynamics to describe the evolution of the system from $k = 0 \rightarrow N$ as

$$\begin{aligned} \boldsymbol{x}(1) &= A(0)\boldsymbol{x}(0) + B(0)\boldsymbol{u}(0) \\ \boldsymbol{x}(2) &= A(1)\boldsymbol{x}(1) + B(1)\boldsymbol{u}(1) \\ &\vdots \\ \boldsymbol{x}(N) &= A(N-1)\boldsymbol{x}(N-1) + B(N-1)\boldsymbol{u}(N-1) \end{aligned} \tag{2.2}$$

Foss and Heirung (2016) considers MPC for a Linear Time Invariant (LTI) system, which gives

$$A(k) = A, \quad \forall \ k$$

Further, as $\boldsymbol{x}(0)$ is a known constant for the optimization, whilst the remaining states and inputs are optimization variables, Foss and Heirung (2016) rearranges (2.2) as

$$\begin{aligned} \boldsymbol{x}(1) - B\boldsymbol{u}(0) &= A\boldsymbol{x}(0) \\ \boldsymbol{x}(2) - A\boldsymbol{x}(1) - B\boldsymbol{u}(1) &= 0 \\ &\vdots \\ \boldsymbol{x}(N) - A\boldsymbol{x}(N-1) - B\boldsymbol{u}(N-1) &= 0 \end{aligned} \tag{2.3}$$

By defining

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}(1) \\ \boldsymbol{x}(2) \\ \vdots \\ \boldsymbol{x}(N) \end{bmatrix}$$

and

$$\boldsymbol{U} = \begin{bmatrix} \boldsymbol{u}(0) \\ \boldsymbol{u}(1) \\ \vdots \\ \boldsymbol{u}(N-1) \end{bmatrix}$$

Foss and Heirung (2016) rewrites (2.3) on matrix form as

$$\underbrace{\begin{bmatrix} \boldsymbol{I} & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} & -\boldsymbol{B} & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} \\ -\boldsymbol{A} & \ddots & \ddots & & \vdots & \boldsymbol{0} & \ddots & \ddots & & \vdots \\ \boldsymbol{0} & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \boldsymbol{0} & \vdots & & \ddots & \ddots & \boldsymbol{0} \\ \boldsymbol{0} & \dots & \boldsymbol{0} & -\boldsymbol{A} & \boldsymbol{I} & \boldsymbol{0} & \dots & \dots & \boldsymbol{0} & -\boldsymbol{B} \end{bmatrix}}_{\boldsymbol{A}_{eq}} \underbrace{\begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{U} \end{bmatrix}}_{\boldsymbol{Z}} = \underbrace{\begin{bmatrix} \boldsymbol{A}\boldsymbol{x}(0) \\ \boldsymbol{0} \\ \vdots \\ \vdots \\ \boldsymbol{0} \end{bmatrix}}_{\boldsymbol{B}_{eq}} \tag{2.4}$$

which gives the equality constraints

$$\boldsymbol{A}_{eq}\boldsymbol{Z} = \boldsymbol{B}_{eq} \tag{2.5}$$

where $\boldsymbol{Z}$ is the optimization variable. Foss and Heirung (2016) describes that as the goal of the controller is to achieve $\boldsymbol{x} \rightarrow 0$ with the minimal use of $\boldsymbol{u}$, the MPC for time a single time step k can be formulated as an optimization problem with a quadratic objective function for LTI systems as follows

$$min \sum_{k=1}^{N} \boldsymbol{x}(k)^T Q \boldsymbol{x}(k) + \boldsymbol{u}(k-1)^T R \boldsymbol{u}(k-1) \tag{2.6}$$

subject to

$$\boldsymbol{x}(k) - B\boldsymbol{u}(k-1) = A\boldsymbol{x}(k-1) \tag{2.7}$$

The weight and cost matrices Q and R are positive semi-definite and positive definite respectively. These matrices consists of decision variables, which can be tuned to achieve the desired behaviour of the MPC. Further, for the entire prediction horizon, Foss and Heirung (2016) proposes the following objective function

$$\begin{aligned} &min \;\; \boldsymbol{Z}^T\boldsymbol{G}\boldsymbol{Z} = \\ &min \quad \begin{bmatrix} \boldsymbol{x}(1) \\ \boldsymbol{x}(2) \\ \vdots \\ \boldsymbol{x}(N) \\ \boldsymbol{u}(0) \\ \boldsymbol{u}(1) \\ \vdots \\ \boldsymbol{u}(N-1) \end{bmatrix}^T \begin{bmatrix} Q & \mathbf{0} & \dots & \dots & \dots & \dots & \dots & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & & & & & \vdots \\ \vdots & \ddots & \ddots & \ddots & & & & \vdots \\ \vdots & & \ddots & Q & \ddots & & & \vdots \\ \vdots & & & \ddots & R & \ddots & & \vdots \\ \vdots & & & & \ddots & \ddots & \ddots & \vdots \\ \vdots & & & & & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \dots & \dots & \dots & \dots & \dots & \mathbf{0} & R \end{bmatrix} \begin{bmatrix} \boldsymbol{x}(1) \\ \boldsymbol{x}(2) \\ \vdots \\ \boldsymbol{x}(N) \\ \boldsymbol{u}(0) \\ \boldsymbol{u}(1) \\ \vdots \\ \boldsymbol{u}(N-1) \end{bmatrix} \end{aligned} \tag{2.8}$$

Foss and Heirung (2016) thus presents the MPC as the following optimization problem

$$min \;\; \boldsymbol{Z}^T\boldsymbol{G}\boldsymbol{Z} \tag{2.9}$$

subject to the equality constraints

$$\boldsymbol{A}_{eq}\boldsymbol{Z} = \boldsymbol{B}_{eq} \tag{2.10}$$

and the inequality constraints

$$\begin{aligned} \boldsymbol{u}_{min} \leq \boldsymbol{u}(k) \leq \boldsymbol{u}_{max}, \quad \forall \;\; k \\ \boldsymbol{x}_{min} \leq \boldsymbol{x}(k) \leq \boldsymbol{x}_{max}, \quad \forall \;\; k \end{aligned} \tag{2.11}$$

The MPC derived in Foss and Heirung (2016) have a quadratic objective function and linear equality and inequality constraints. This implies that the MPC is formulated as a convex Quadratic Programming (QP)-problem, and can thus be solved using QP-solvers. This is an advantage over Nonlinear Programming (NLP)-problems as QP-solvers are proven to be efficient, which might give a reduced calculation time compared to a more complex NLP-solvers.

Further, Foss and Heirung (2016) discuss the weaknesses of the MPC, and describes that there are two main weaknesses. These are described as infeasiblity and lack of stability for the MPC algorithm. Infeasibility occurs when there does not exist any feasible solutions to the optimization problem, e.g

$$\boldsymbol{x}(k+1) > \boldsymbol{x}_{max}, \quad \forall \;\; \boldsymbol{u}_{min} \leq \boldsymbol{u}(k) \leq \boldsymbol{u}_{max}$$

as the solver will not be able to find a solution to this problem without violating any input constraints. Foss and Heirung (2016) proposes a solution to ensure feasibility by introducing slack variables $\boldsymbol{\epsilon}$, such that

$$\boldsymbol{x}_{min} - \boldsymbol{\epsilon} \leq \boldsymbol{x} \leq \boldsymbol{x}_{max} + \boldsymbol{\epsilon}$$

as this will soften the state constraints. By allowing $\boldsymbol{\epsilon}$ to be large enough, a feasible solution may always exist to the MPC.

The other problem with the use of MPC is that stability may not be guaranteed for a system even if feasible solutions exists for every time step. Foss and Heirung (2016) does not go in depth on stability for MPC, however one solution is proposed. By enforcing the constraint $\boldsymbol{x}(N) = \mathbf{0}$, convergence to zero, and thus stability is achieved provided a feasible solution exists. This may however affect the feasibility of the MPC, as one may not ensure feasibility for every time step by including slack variables whilst simultaneously enforcing $\boldsymbol{x}(N) = \mathbf{0}$.

2.1.2 Nonlinear Model Predictive Control

The theory of Nonlinear Model Predictive Control (NMPC) is described in Grune and Pannek (2017), to be an open loop optimal control law. The NMPC is similar to MPC, but the main difference is that the NMPC handles nonlinear constraints, and can thus be implemented on systems with nonlinear dynamics. As the NMPC uses NLP solvers to solve the algorithm, one might also implement nonlinear objective functions in the control law. However, as stated in Grune and Pannek (2017) the quadratic objective function, as stated in (2.8) is still a popular choice for NMPC, as this aims to converge the states and inputs towards zero.

Further, Grune and Pannek (2017) describes the computational complexity of the NMPC control law, as well as the NLP solvers commonly used to solve these problems. One of the solvers described in Grune and Pannek (2017) is the Sequential Quadratic Programming (SQP) solver, which is also described in Nocedal and Wright (2006).

The SQP solver is an Active Set method, which searches for the optimal solution by considering a set of the active constraints, calculating the lagrange multipliers for these constraints, eliminating a subset of the constraints with negative lagrange multipliers, and searching for infeasible solutions. This approach is repeated until the solution is within a tolerance. Both Grune and Pannek (2017) and Nocedal and Wright (2006) describes that a good initial value in the search of an optimal solution, i.e the value the algorithm starts the search from, may significantly reduce the computation time of the SQP algorithm. Thus, Grune and Pannek (2017) and Nocedal and Wright (2006) argues that the optimal solution found in the previous iteration may be used as the initial point for the next iteration, i.e the optimal solution in iteration k will serve as the initial point in the search for a new solution in iteration $k + 1$. As the change of the optimal solution may be relatively small between iterations, this choice of initial position might be close to the optimal solution, which might significantly reduce the computation time the algorithm uses to find a new optimal solution. This approach is referred to as a "warm start" for the algorithm, were the optimal solution found for the prediction horizon in time step k is used as the initial value for the search in time step $k + 1$, as this provides initial values close to the optimal value and may thus reduce the computation time.

2.1.3 Non-conventional approaches to Model Predictive Control

Multiplexed Model Predictive Control

One study which proposes a solution to the problem with the MPC's large computational time for control systems with many control variables is the use of Multiplexed Model Predictive Control (MMPC). As described in Ling et al. (2011), the principle behind MMPC is to divide the system into smaller subsystems which the controller will handle in order. The idea behind this is that the algorithm will calculate and apply the input for one subsystem before moving on to the next. The motivation behind this is that the controller will apply the control input faster for each subsystem as each control input is calculated separately, and thus reduce the computation time. According to Ling et al. (2011), a standard MPC has a computation time of $O((m \times N)^3)$, where m is the number of states and N is the prediction horizon length. When using the MMPC approach, each of the states are handled separately and by reducing the number of states for each subsystem, the computation time will be reduced drastically.

An example made by Ling et al. (2011) describes a finite horizon control problem with four mass-spring dampers coupled together, where the control objective was to minimize the control energy of the system. This problem was solved using both a standard MPC and a MMPC, and simulation results showed that for shorter horizon lengths the standard MPC had shorter computation time, however as the horizon length was increased the MMMPC showed a significant improvement in terms of computation time. The performance of the MMPC and the regular MPC is however not shown for comparison in the article, and we can therefore not comment on the performance of the controller. One problem with the MMPC approach is that dividing the system into subsystems offers sub-optimal control inputs for each time step as dependencies between some states and inputs are removed. However Ling et al. (2011) argues that the reduced calculation time for each control input compensates for the sub-optimal input as it is often more important to achieve a good control input with a faster response than a better control input with a much slower response. The implementation of MMPC therefore seems to be a good approach for large system with great computational cost where the states are heavily affected by time varying disturbances, as the reduced computation time gives a faster response to the change of disturbance. The weakness of the MMPC however seems to be for systems with lower time horizon, whilst many of the states are correlated and where the effect of external disturbances are negligible.

Simultaneous Model Predictive Control and state estimation

There are some studies which uses a MPC for an output feedback law for a control system. One study done on this topic is described in Copp and Hespanha (2016), which proposes a control law which simultaneously estimates the states as well as computing the optimal input sequence for each time step. This approach is proposed for systems with unmeasured states where the measurement is affected by noise. The idea behind this control law is to calculate an optimal control input which minimizes the cost function, as for a normal MPC, however the cost function now also includes compensation for the noise affecting the states and measurements. The algorithm also includes a Moving Horizon Estimator

(MHE), which use a similar optimization algorithm as the MPC to predict future states based on the past states.

2.2 Path following control

2.2.1 Kinematic path following controller

Siciliano and Khatib (2008) describes a path following algorithm for a car-like vehicle in 3-DOF. The vehicle have the three following states $(x^b \;\; y^b \;\; \psi)^T$. Further the robot is assumed to be nonholonomic, meaning it is constrained such that it can not move directly sideways, i.e $\dot{y}_b = 0$. The vehicle is controlled by the input vector $\mathbf{u} = [u_1 \;\; u_2]^T$, where u_1 is the forward velocity in BODY-frame and u_2 is the steering angle.

The proposed control law uses a feedback linearizing controller to drive the vehicle to the desired path. Further global asymptotical stability i shown for this path following controller by the use of Lyapunov functions. This path following controller is solely based on the vehicle's kinematics, and does not consider any kinetics. Siciliano and Khatib (2008) justifies this choice as the low-level controllers are assumed good enough to keep the error in desired velocity and heading angle low, even if the vehicle dynamics are not considered. Further, kinematic models are less complex than kinetic models, and that the simplicity of the kinematic controller outweighs the precision of the kinetic controller.

2.2.2 Kinetic path following control

A control law for a kinetic nonlinear vehicle model is considered in Mashadi et al. (2014). Mashadi et al. (2014) defines that the vehicle has a constant velocity, and that the control objective is to converge y_e and ψ_e towards zero, where y_e and ψ_e are the BODY-fixed lateral error and heading errors respectively. By converging y_e towards zero, Mashadi et al. (2014) ensures that the vehicle converges to the path, and further, by converging ψ_e towards zero, the vehicle will achieve the desired heading to reach the next path position. By linearization of the nonlinear vehicle dynamics, y_e and ψ_e and their derivatives are presented on state space form. Further, uncertainty terms are introduced in the model to compensate for modelling errors, and the state space model is used to derive a control law. Simulation of the control law shows that it is able to follow the desired path.

2.2.3 MPC for path following problems

Another way of solving path following problems is by the use of MPC. One approach to this is described in Yu et al. (2011), which proposes a NMPC formulation to solve a path following problem for a car-like mobile robot, similar to the one described in 2.2.1. The control objective is that the robot should follow a reference path that is twice continuously differentiable, which is also an requirement for the proposed control law to be valid. The approach in Yu et al. (2011) proposes a problem formulation for the path following objective, where the vehicle kinematics are described as a NMPC problem, and by the use of Lyapunov stability as the cost function, derives a controller that ensures asymptotic convergence to the desired path. Lastly the proposed NMPC was simulated compared to

a Line of Sight (LOS) controller for a car following an eight-shaped path. The results presented in Yu et al. (2011) shows better performance for the NMPC than for the LOS control law, as the NMPC has faster convergence towards the path for all starting points. Both controllers are able to follow the desired path after convergence.

Mata et al. (2017) describes a different approach to solving path following problems using MPC. Mata et al. (2017) describes an approach where a MPC formulation is derived for a LTV model of a bicycle robot. The control objective for the bicycle robot is to reduce the body fixed lateral error y_e^b and the orientation error ψ_e according to a predefined path. The vehicle dynamics are described on state space form, i.e $\dot{\boldsymbol{x}} = \mathbf{A}(t)\boldsymbol{x}(t) + \mathbf{B}(t)u(t)$, with $\boldsymbol{x} = [\boldsymbol{y} \;\; \dot{\boldsymbol{y}} \;\; \boldsymbol{\psi} \;\; \dot{\boldsymbol{\psi}}]^T$ and $u(t)$ is the steering angle δ. The control law is based on a quadratic cost function for every time step, with an introduced separate penalty function for the final time step. This penalty function on the final time step is to ensure that the final time step will have reached the desired waypoint. As the vehicle model is time variant, the model is updated for every time step of the MPC algorithm.

In order to guarantee stability for the MPC, Mata et al. (2017) proposes the use of a second controller in addition to the MPC. The motivation behind this is that the MPC is no longer required to ensure convergence to zero, but rather it should ensure convergence to an invariant set Ω containing the equilibrium. When the system has converged to Ω, a local controller can be used to further ensure convergence to the equilibrium. The system described in Mata et al. (2017) uses a Linear Quadratic Regulator (LQR) to ensure stability once the system has converged to the invariant set.

Yu et al. (2011) and Mata et al. (2017) describe different approaches to the choice of the reference path. The reference path $\boldsymbol{r}$ used in Yu et al. (2011) is parameterizied by a scalar value s, such that

$$\boldsymbol{r} = \boldsymbol{p}(s) \tag{2.12}$$

where the function $\boldsymbol{p}$ is twice continuously differentiable. The time evoulution of $s(t)$ is not necessary to be known prior to the application, but rather is affected by a virtual control input $v(t)$, such that

$$\dot{s}(t) = v(t) \tag{2.13}$$

where the virtual control input $v(t)$ is calculated for future time steps by the NMPC algorithm, and thus ensuring a smooth evolution of the reference path. The reference path in Mata et al. (2017) however, is described as a series of waypoint coordinates defined in the inertial frame. As the longitudinal velocity is constant for the entire prediction horizon, the reference path is described as straight line segments between the predefined waypoints, i.e between the waypoints r_1 and r_2 we have a vector of reference points $\boldsymbol{r}_{12} = [r_1 + hu,\, r_1 + 2hu,\, ...,\, r_2]$, where h is the step time and u is the body-fixed longitudinal velocity.

Yu et al. (2011) also argues that as the system is nonlinear and non-convex, one can not guarantee to obtain the global optimal solution to the control problem, and that a locally optimal solution should be applied instead. This is further discussed in Zhang et al. (2015), which considers the issue if no solution is found within the required time interval. Zhang et al. (2015) proposes three different approaches to solve this problem. The first approach is to force a time constraint for the calculation, where the search for an optimal solution is canceled after the desired time interval, and the best solution so far is applied whether it is

the optimal input or not. This approach however faces the risk that no feasible solution is found within the time limit.

Another way to enforce the time constraint is by the use of several optimization engines, where the first to return a feasible solution is selected. The second approach is to operate in degraded mode if no solution is found. This can be done in several ways, for example by applying the optimal solution calculated in the previous time step if no solution is found for this time step. The weakness of this approach however is that it does not account for any disturbances that might have affected the system between the time steps. The effect of these disturbances might increase significantly if the system takes several time steps before a new solution is found.

The third approach is to select a less complex model for the implementation of the MPC, as the reduced complexity might significantly reduce the calculation time for the algorithm. The problem with this approach however is that some dynamics might get lost by applying less complex models, e.g a linearized model might only be valid around the point of linearization, which will affect the performance of the controller. In addition to these approaches, Mata et al. (2017) proposes an approach where the control horizon and prediction horizon is chosen separately, such that the control input is calculated for N time steps, whilst the future states are calculated for N+K time steps. Using this approach the control input at time step N will be held constant for the prediction of the next K time steps.

Chapter 3

Basic Theory

3.1 Vehicle Model

In order to derive a MPC control law for the UGV, a model has to be derived. For this thesis, two types of models will be considered, i.e kinematic and kinetic models. Kinematics is defined as the study of motion without regarding the cause of said motion, whilst kinetics is defined as the study of motion and its cause. The minimum requirement of our controller is to be able to follow a path in 3-DOF, and we would thus at least require a kinematic model of the vehicle for $[x^b \;\; y^b \;\; \psi]^T$. However, as the actual UGV will be moving in a three dimensional (3D) space, a model in 6-DOF is required to accurately describe the UGV's motion. Siciliano and Khatib (2008) argues that kinematic models are sufficient for most path following controllers, however by introducing kinetic models, one might predict the change of motion, and thus increase the accuracy.

In the field of kinematics, one studies the motion of objects with regards to different reference frames. For FFI's UGV, OLAV, the position i measured in the ENU-frame. In Dyrnes (2018), a new coordinate frame, the Wheel Contact Point (WCP)-frame, was defined as a plane where all the wheels touch the ground. This is different from the commonly used BODY-frame, because of the suspension from the wheels to the vehicle, which might cause the vehicle's body to be oriented be differently than that of the contact points with the ground. The WCP-frame was thus introduced as the vehicle's propulsion is applied through the contact points between the wheels and ground, and thus the orientation of the wheels might relevant to describe the motion.

A kinematic model for the UGV was derived in Dyrnes (2018), where OLAV's motion,

or rather the motion of OLAV's WCP-frame, was expressed in ENU-frame as

$$\begin{bmatrix} \dot{\boldsymbol{p}}^e_{w/e} \\ \dot{\boldsymbol{\Theta}}_{ew} \end{bmatrix} = \begin{bmatrix} u\,c(\psi)c(\theta_w) + v\,(c(\psi)s(\theta_w)s(\phi_w) - c(\phi_w)s(\psi)) + w\,(s(\psi)s(\phi) + c(\psi)c(\phi)s(\theta)) \\ u\,c(\theta_w)s(\psi) + v\,(c(\psi)c(\phi_w) + s(\psi)s(\theta_w)s(\phi_w) + w\,(c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi)) \\ -u\,s(\theta_w) + v\,c(\theta_w)s(\phi_w) + w\,c(\theta)c(\phi) \\ p_w + q_w\,s(\phi_w)t(\theta_w) + r_w\,c(\phi_w)t(\theta_w) \\ q_w\,c(\phi_w) - r_w\,s(\phi_w) \\ q_w\,\frac{s(\phi_w)}{c(\theta_w)} + r_w\,\frac{c(\phi_w)}{c(\theta_w)} \end{bmatrix} \tag{3.1}$$

Further, Dyrnes (2018) argues that $v \approx 0$ and $w \approx 0$, and that the kinematic model thus simplifies to

$$\begin{bmatrix} \dot{\boldsymbol{p}}^e_{w/e} \\ \dot{\boldsymbol{\Theta}}_{ew} \end{bmatrix} = \begin{bmatrix} \dot{x}^e \\ \dot{y}^e \\ \dot{z}^e \\ \dot{\phi}_w \\ \dot{\theta}_w \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} u\,c(\psi)c(\theta_w) \\ u\,c(\theta_w)s(\psi) \\ -u\,s(\theta_w) \\ p_w + q_w\,s(\phi_w)t(\theta_w) + r_w\,c(\phi_w)t(\theta_w) \\ q_w\,c(\phi_w) - r_w\,s(\phi_w) \\ q_w\,\frac{s(\phi_w)}{c(\theta_w)} + r_w\,\frac{c(\phi_w)}{c(\theta_w)} \end{bmatrix} \tag{3.2}$$

In the field of kinetics, one studies the cause of a objects motion. As OLAV is not actuated in the y^b and z^b directions, Dyrnes (2018) models the forces affecting the motion in these directions as disturbances. The pitch and roll angle, θ and ϕ is not actuated either, however Dyrnes (2018) derives that the forces affecting these motions are caused by the orientation of the terrain as OLAV is moving in x^b direction, and a kinetic model for the motion of the WCP-frame is thus derived as

$$\begin{bmatrix} \dot{\boldsymbol{\nu}}^w_{w/e} \\ \dot{\boldsymbol{\omega}}^w_{w/e} \end{bmatrix} = \begin{bmatrix} \dot{u} \\ \dot{v} \\ \dot{w} \\ \dot{p}_w \\ \dot{q}_w \\ \dot{r}_w \end{bmatrix} = \begin{bmatrix} \frac{1}{m_v}[r_{cg}(F_{fw,x}\cos(\gamma) + F_{rw,x}) + F_{g,x} + F_d] \\ \frac{F_y}{m_v} \\ \frac{F_z}{m_v} \\ \frac{\frac{\dot{u}}{W}(\sin(\theta_{g,l}) - \sin(\theta_{g,r})) + \frac{u}{W}(\cos(\theta_{g,l})\dot{\theta}_{g,l} - \cos(\theta_{g,ri})\dot{\theta}_{g,ri})}{\sqrt{1-(\frac{u}{W}(\sin(\theta_{g,l}) - \sin(\theta_{g,ri})))^2}} \\ \frac{\dot{u}}{A_s}(\theta_{g,f} - \theta_{g,r}) + \frac{u}{A_s}(\dot{\theta}_{g,f} - \dot{\theta}_{g,r}) \\ \frac{1}{I_z}(u|u|\mu_f\sin(\gamma)R - Dr) \end{bmatrix} \tag{3.3}$$

where $\theta_{g,r}$ and $\theta_{g,f}$ denotes the inclination of the ground at the rear and front wheels respectively. $\theta_{g,l}$ and $\theta_{g,ri}$ denotes the inclination of the left and right wheels respectively. For the yaw dynamics, a nonlinear damping model was also proposed. We therefore have two models that are considered for the yaw dynamics, the linear and nonlinear damping model, which are given in (3.4) and (3.5)

$$\dot{r}_w = \frac{1}{I_z}(u|u|\mu_f\sin(\gamma)R - Dr) \tag{3.4}$$

$$\dot{r}_w = \frac{1}{I_z}(u|u|\mu_f\sin(\gamma)R - Dr|r|) \tag{3.5}$$

It is important to note here that these models describe the motion of the WCP-frame, however this frame can not be directly measured as this is a theoretical plane and not a physical part of the vehicle. The implementation of the WCP-frame is thus not trivial, however with a good kinetic model describing the suspension of the UGV, an estimation can be made and a control law for the WCP-frame can be implemented.

3.2 Testing the vehicle model

The vehicle model was tested in 3-DOF in Dyrnes (2018), and the results are shown in **Fig. 3.1** and **Fig. 3.2**. As we can see, the models have different performances. Dyrnes (2018) summarizes that the linear model seems to be more accurate for higher velocities, whilst the nonlinear model seems to be more accurate for lower velocities. It is at this point worth mentioning a mistake made in Dyrnes (2018). The measurements was made with several heavy sensors mounted on the roof of OLAV, however the model was derived for OLAV without these mounted sensors. This may thus have had an impact on the accuracy of the simulations, as the heavy sensors would have both affected the moment of inertia of the vehicle, as well as off-setting the Centre of Gravity (CG). Some new measurements, without the mounted sensors, are thus in order to test the model with more accuracy.

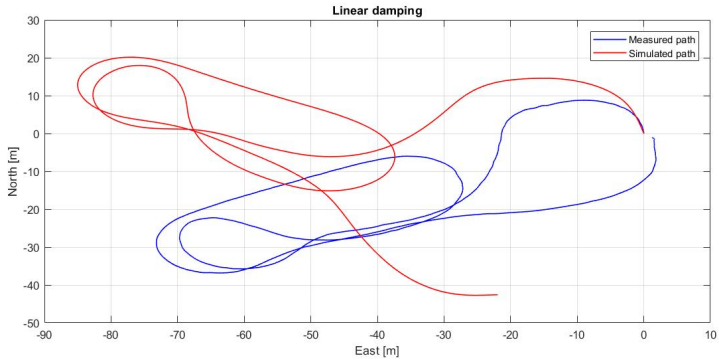

Figure 3.1: Comparison of measured position and simulated position with linear damping model from Dyrnes (2018)

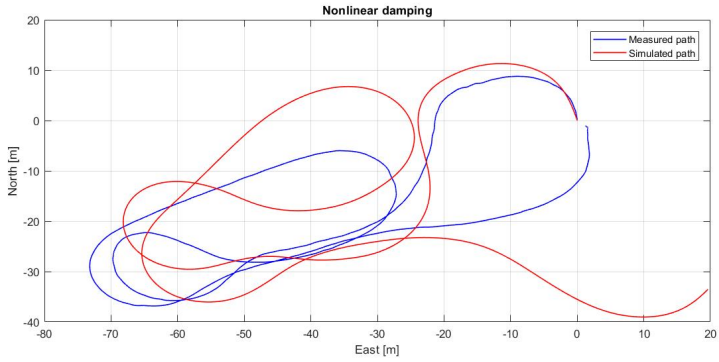

Figure 3.2: Comparison of measured position and simulated position with nonlinear damping model from Dyrnes (2018)

3.3 Model Parameters

3.3.1 Moment of inertia

OLAV's moment of inertia was estimated in Dyrnes (2018) as $I_z = 1050\text{kgm}^2$. However, these were not made using measurements of the actual vehicle, but rather by measuring a scaled model in the UGV's specification sheet. Therefore, in order to improve the accuracy of the moment of inertia, we will make new estimations using measurements from the actual vehicle. The calculation will still be made using the formula derived in Dyrnes (2018), which is shown in (3.6) and (3.7). Dyrnes (2018) defined that the vehicle could be partitioned as shown in **Fig. 3.3** where the x, y and z values will be the distances from the UGV's CG. Further, the total inertia can be found by summing the inertia for all the partitions.

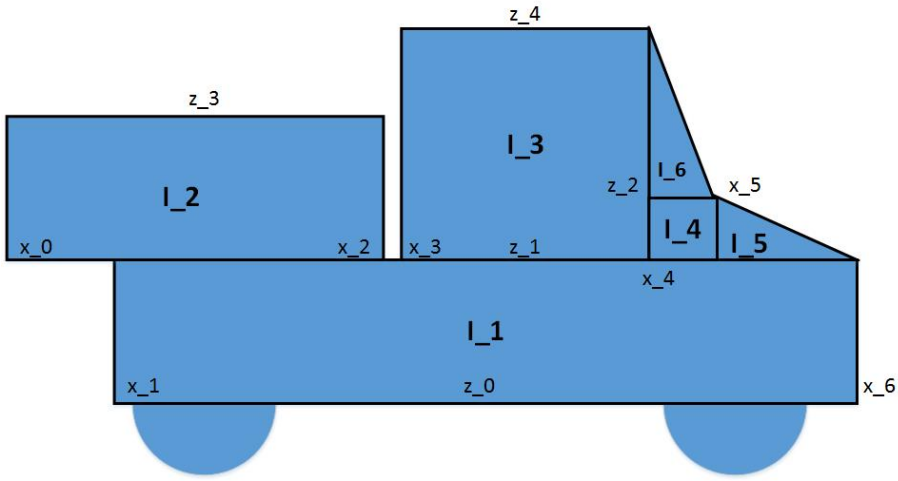

Figure 3.3: Vehicle partition in xz-plane from Dyrnes (2018) (not to scale)

Dyrnes (2018) derived that the inertia for the cubic partitions can be calculated as follows

$$I_{z,n} = \rho_m \frac{1}{3}(z_{max} - z_{min})((x_{max}^3 - x_{min}^3)(y_{max} - y_{min}) + (y_{max}^3 - y_{min}^3)(x_{max} - x_{min})) \tag{3.6}$$

where ρ_m is the density, whilst the inertia for the prismatic partitions can be calculated by using the following formulae

$$\begin{aligned} I_{z,n} = \rho_m(a(&\frac{1}{4}(x_{min}^4 - x_{max}^4) + \frac{x_{min}}{3}(x_{max}^3 - x_{min}^3)) \\ &+ \frac{z_{max} - z_{min}}{3}(x_{max}^3 - x_{min}^3))(y_{max} - y_{min}) \\ &+ \frac{1}{3}(y_{max}^3 - y_{min}^3)((z_{max} - z_{min})(x_{max} - x_{min}) \\ &+ a(x_{min}(x_{max} - x_{min}) - \frac{1}{2}(x_{max}^2 - x_{min}^2))) \end{aligned} \tag{3.7}$$

where

$$a = \frac{z_{min} - z_{max}}{x_{max} - x_{min}} \tag{3.8}$$

The measurements of the vehicle is shown in **Tab. 3.1**. The density of the UGV is found by dividing the mass of the vehicle on the total volume. In reality, ρ_m will not be equal for the different pieces, however this simplified approximation is used as it is impossible to measure the weight and calculate density for the different partitions.

Symbol		Value
x_0	=	-1.43m
x_1	=	-1.09m
x_2	=	-0.41m
x_3	=	-0.22m
x_4	=	0.92m
x_5	=	1.29m
x_6	=	1.72m
z_0	=	-0.30m
z_1	=	0.275m
z_2	=	0.595m
z_3	=	0.875m
z_4	=	1.31m
y_0	=	-0.78m
y_1	=	0.78m

Table 3.1: Numerical values for partitions

The volume is calculated for each partition using the values from **Tab. 3.1**, and the total volume is the sum of the partitions. We thus have

$$\rho_m = \frac{m}{V} = \frac{1200\text{kg}}{5.8143\text{m}^3} = 206\text{kg/m}^3 \tag{3.9}$$

which gives

$$I_z = \rho_m \sum_{i=1}^{6} = \rho_m(2.4198{+}1.0845{+}0.2651{+}0.1768{+}0.5486{+}0.7120)\text{kgm}^2 = 1075\text{kgm}^2 \tag{3.10}$$

3.3.2 Damping and friction coefficients

As the damping and friction coefficients D and μ_f found in Dyrnes (2018) were based on trial and error, they might be inaccurate for any other path than the one tested in **Fig. 3.1** and **Fig. 3.2**. In order to get a more accurate model for the the controller, we will therefore determine this coefficients with more accuracy.

Before determining the coefficients, we will find the transfer function describing the yaw dynamics. The linear yaw dynamics are given in (3.3) and can be rearranged as follows

$$\begin{aligned} I_z\dot{r} &= u|u|\mu_f R\sin\gamma - Dr \\ \implies I_z\dot{r} + Dr &= u|u|\mu_f R\sin\gamma \end{aligned} \tag{3.11}$$

As $u|u|\sin\gamma$ is the applied torque to the system whilst μ_f and R are system coefficients, we can substitue the input with an input variable $\tau = u|u|\sin\gamma$. Further, by performing a Laplace transformation, we get the transfer function as

$$\frac{r(s)}{\tau(s)} = \frac{\mu_f R}{I_z s + D} = \frac{\frac{\mu_f R}{D}}{\frac{I_z}{D}s + 1} = \frac{K}{Ts + 1} \tag{3.12}$$

The K and T can be identified by performing a step response on the system. This was done by driving OLAV on a flat area without obstacles with approximately constant velocity, before turning the wheel whilst maintaining constant velocity. As turning the wheel causes a change of the steering angle γ, and assuming u maintains approximately constant during the turning, this will cause a step in τ. In reality, γ can not change momentarily, however assuming the dynamic of γ is fast enough, this dynamic can be disregarded. The recorded step response for r is shown in **Fig. 3.4**.

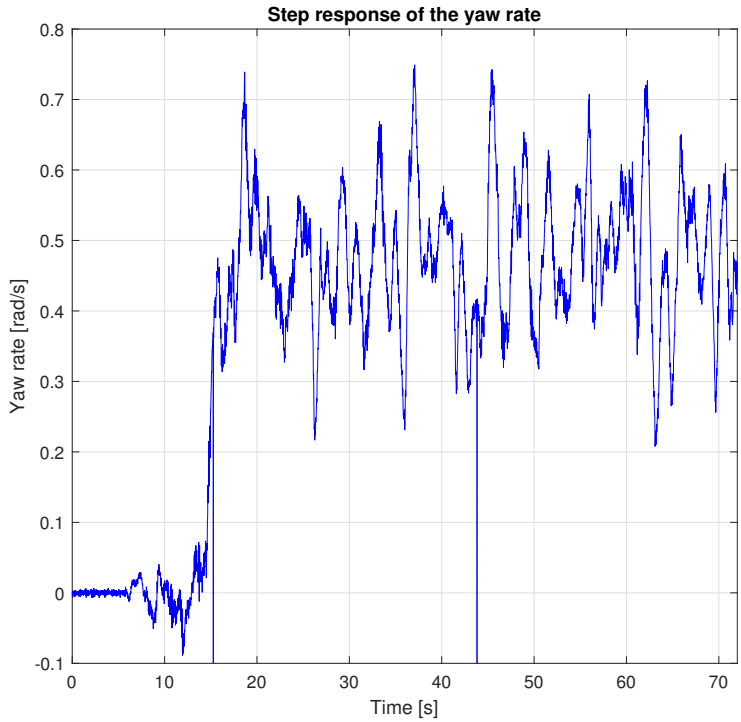

Figure 3.4: Step response of the yaw rate

As we can see, this step response is affected by disturbances. These disturbances are caused by measurement noise, as well as small changes in u and γ due to human error. By using the "mean" function in MATLAB, we find the mean stationary value to be $r = 0.472\mathrm{rad/s}$. Further, an attempt at curve fitting gives the approximated step response as in **Fig. 3.5**.

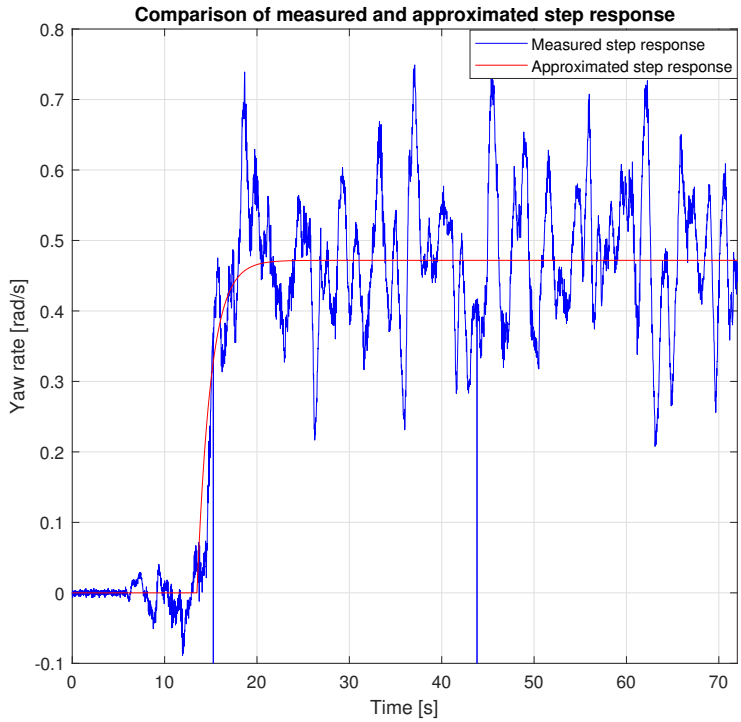

Figure 3.5: Measured step response compared to an approximated step response

We can thus use this approximation find the parameters of the transfer function. We will first find the time constant T, which is defined as the time it takes the system to reach 63.2% of its stationary value, i.e $t \to \infty$. As stated earliear, the stationary value is 0.472, and thus the time constant is the time it takes to reach $r = 0.472 \cdot 0.632 = 0.316$. This is illustrated in **Fig. 3.6**

And thus we can calculate

$$T = 14.99 - 13.50 = 1.49 \tag{3.13}$$

and further, by using (3.12), we can estimate the damping coefficient as

$$D = \frac{I_z}{T} = \frac{1075\text{kgm}^2}{1.49\text{s}} = 721\text{kgm}^2/\text{s} \tag{3.14}$$

The friction coefficient can be found by considering the stationary value as we have

$$s \to 0 \quad \Longrightarrow \quad r = \frac{\mu_f R}{D}\tau = \frac{\mu_f R}{D} u|u| \sin\gamma \tag{3.15}$$

Thus, by rearranging we have

$$\mu_f = \frac{D\,r}{Ru|u|\sin\gamma} = \frac{721 \cdot 0.472}{1.26 \cdot 4.76 \cdot |4.76| \cdot \sin 0.32} = 37.9 \tag{3.16}$$

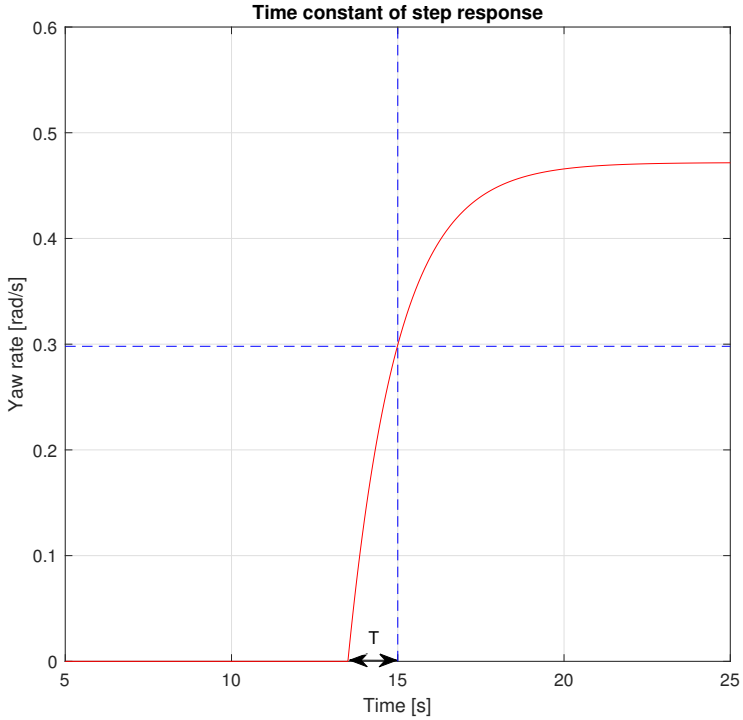

Figure 3.6: Time constant for the transfer function

An approximation of the nonlinear damping coefficient is found by assuming that the turning rate has reached the stationary value at $r = 0.472$, i.e $\dot{r} = 0$. By using the friction coefficient found for (3.16), we have

$$\begin{aligned} I_z\dot{r} &= R\mu_f u|u| \sin\gamma - Dr|r| = 0 \\ \implies D\, r|r| &= R\mu_f u|u| \sin\gamma \\ \implies D &= \frac{R\mu_f u|u| \sin\gamma}{r|r|} \\ &= \frac{1.26 \cdot 37.9 \cdot 4.76 \cdot |4.76| \cdot \sin 0.32}{0.472 \cdot |0.472|} = 1528 \end{aligned} \tag{3.17}$$

Chapter 4

MPC

4.1 Reference path

The goal of the control laws of this thesis is to drive the UGV towards a desired path $\mathbf{\Lambda}$, before following this path towards the end point. For the controllers of this thesis, the positions will be defined in a local reference frame $\boldsymbol{p}^l$, which is aligned with the ENU-frame, however the positions will be given in meters from origo rather than degrees. The origo of this local frame will be defined as the final waypoint, as this simplifies the generation of paths. FFI is currently using UGV paths that are defined by recorded waypoints as well as a desired steering angle between said waypoints and a nominal velocity. Given that the number of waypoints in the reference path is equal to the number of time steps required to follow this path, e.g by recording a path when driving OLAV manually, and that the UGV starts at the desired path, we could use the recorded waypoints as reference positions $\boldsymbol{p}_d^e$ for the MPC. Thus we could aim to follow the path by minimizing $\boldsymbol{p}_d^l(k) - \boldsymbol{p}^l(k)$ for every time step. However, this might cause a problem if the system is prone to modelling errors or disturbances, or if OLAV does not start close to a desired waypoint, as the errors might amplify for later time steps. An example of this is if $\boldsymbol{p}^l(0) \neq \boldsymbol{p}_d^l(0)$, with a sufficiently large error, no control input within the physical constraints of the UGV is able to ensure $\boldsymbol{p}^l(1) = \boldsymbol{p}_d^l(1)$. This would cause problems for the MPC, as there is no feasible solution to the control problem. Further, as it is not a requirement for this system to follow the path in a given time, the desired waypoint at time t can thus be chosen freely as long as the controller follows the desired path. It is therefore desirable to derive a reference positions that will follow the desired path, without requiring that a waypoint is achieved at a certain time step. We will therefore define the paths differently for this thesis, than FFI's current definition. This is done as a new definition of the path will serve the optimization formulation of the MPC control laws derived in this thesis.

As the UGV may not follow the path with the same velocity as the path was recorded, the distance covered in each time step might vary from the recorded path. In order to ensure that the UGV always has a reference position along this path, we will generate continuous smooth path $\mathbf{\Lambda}$ which contains all the recorded waypoints. This path will consist of the

recorded positions x^e and y^e converted to the local frame, i.e x^l and y^l, and we thus have $\Lambda = [\lambda_x \ \ \lambda_y]^T$, where λ_x and λ_y are the reference paths in x and y direction respectively. As this path is continuous, it can defined as a continuous function of some parameter s, i.e

$$\Lambda(s) = \begin{bmatrix} \lambda_x(s) \\ \lambda_y(s) \end{bmatrix} = \begin{bmatrix} f_1(s) \\ f_2(s) \end{bmatrix}, \qquad s \in [0, \ n] \tag{4.1}$$

where n and 0 are the first and last waypoints in the desired path respectively, as $s = 0$ being the final waypoint will simplify the optimization of the control law. It is beyond the scope of this thesis to go in depth on path generation, but for the purpose of testing the control law, a couple of paths will be generated. These will be smooth paths in the local xy-plane parameterized by s which satisfies (4.1). The first path we will use for testing, henceforth called $\mathbf{\Lambda}_1$ will be a straight line in the xy-plane, which can be given as

$$\Lambda(s) = \begin{bmatrix} \lambda_x(s) \\ \lambda_y(s) \end{bmatrix} = \begin{bmatrix} a_1 s \\ a_2 s \end{bmatrix} \tag{4.2}$$

The second path, $\mathbf{\Lambda}_2$, will be a half period of a sine wave, which can be given as

$$\Lambda(s) = \begin{bmatrix} \lambda_x(s) \\ \lambda_y(s) \end{bmatrix} = \begin{bmatrix} a_1 s \\ a_2 \sin \omega s \end{bmatrix} \tag{4.3}$$

Even though we have four states for the 3-DOF system, i.e $[x^e \ \ y^e \ \ \psi \ \ \dot{\psi}]^T$, we will only follow a reference path for x^e and y^e. The reason for this is that ψ and $\dot{\psi}$ should be controlled such that ψ will be the required heading to achieve the desired $\dot{x}^e$ and $\dot{y}^e$, rather than achieving a predefined desired heading ψ_d. Further r will be the yaw rate required to achieve the necessary heading, and will not be controlled towards a predefined yaw rate r_d. For the 6-DOF controllers, we will not aim to control z, θ or ϕ as these will be bound by the terrain, and we will thus not define a path for these states.

4.2 Optimization formulation

In general, the control objective for a path following problem is to follow a reference path at all times, i.e $\boldsymbol{p}_d^e(t) - \boldsymbol{p}^e(t) = 0$ as $t \to \infty$. For the MPC, however, the control objective will be to drive the discrete states towards the reference for every time step, i.e, $\boldsymbol{p}_d^e(k) - \boldsymbol{p}^e(k) = 0, \quad k = 1 \to N$, where N is the prediction horizon of the MPC. Further we will simplify the notation by defining $\boldsymbol{P}^e = [\boldsymbol{p}^e(1) \ \ \boldsymbol{p}^e(2) \ \ \dots \ \ \boldsymbol{p}^e(N)]^T$, $\boldsymbol{P}_d^e = [\boldsymbol{p}_d^e(1) \ \ \boldsymbol{p}_d^e(2) \ \ \dots \ \ \boldsymbol{p}_d^e(N)]^T$ and $\boldsymbol{\mathcal{T}} = [\boldsymbol{\tau}(0) \ \ \boldsymbol{\tau}(2) \ \ \dots \ \ \boldsymbol{\tau}(N-1)]^T$ where $\boldsymbol{\tau}$ is the control vector for each time step.

The purpose of the MPC control law is to solve an optimization problem which will yield the input vector for the prediction horizon, $\boldsymbol{\mathcal{T}}$, required to achieve the predicted $\boldsymbol{P}^e$ such that $\boldsymbol{P}_d^e - \boldsymbol{P}^e = 0$. The optimization will be solved with regards to the system dynamics as constraints, e.g $\boldsymbol{p}^N = f(\boldsymbol{p}^e(N-1), \boldsymbol{\tau}(N-1))$, as well as physical constraints affecting the inputs, e.g $\tau \leq \tau_{max}$. For this thesis a variety of MPC approaches will be derived in order to achieve this. The different approaches are expected to have varying effect on accuracy as well as computational cost, however the control objective will remain the same for all approaches, and all approaches will be based on the aforementioned principles.

The basic theory of MPC is based on Foss and Heirung (2016), however we will expand upon this further. First we will introduce a change of notation. The general form of the MPC is given with the system states $\boldsymbol{x}$ and the inputs $\boldsymbol{u}$, whilst for our path following MPC, we will have the states $\boldsymbol{p}^e$ and the inputs $\boldsymbol{\tau}$. We will now derive a new variation of the MPC control law, which will ensure convergence to the path from any starting position, as well as following the path towards the end.

The reference path will be defined as $\boldsymbol{\Lambda}$, and is derived in **4.1**, whilst the vehicle's position for time step k is defined as $\boldsymbol{p}^e$. Similar to Yu et al. (2011), we want the vehicle to converge to the path, before following the path towards the end point. However, Yu et al. (2011) defines the path as twice continuously differentiable, but this will not be a requirement for our control law. We will instead choose the parameter s as a free variable, which can be chosen during the control algorithm. Our main motivation for this choice is that this will give the advantage of choosing the optimal reference position, $\boldsymbol{p}_d^e$ as well as the optimal input to reach this position.

Lets consider a simple generalized example where a path is defined as

$$\lambda_x(s) = 0.5s \tag{4.4a}$$

$$\lambda_y(s) = 0.2s \tag{4.4b}$$

$$s \in [0,\ 20] \tag{4.4c}$$

and the initial position is $\boldsymbol{p}^e(0) = [2\ \ 3]^T$. For this simple example we will only consider a point in a two dimensional space, and the orientation i.e ψ will not be considered. If s is chosen as 0, the reference position is $\boldsymbol{p}_d^e = [0\ \ 0]^T$ and the initial error is 3.61. This is illustrated in **Fig. 4.1**

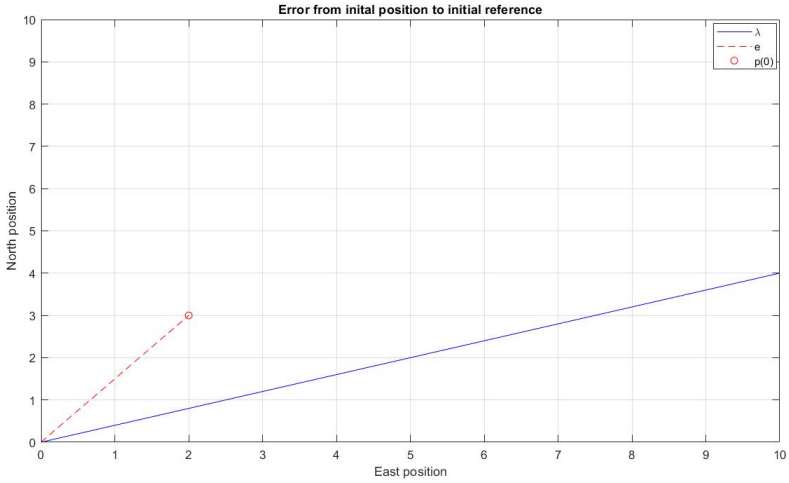

Figure 4.1: Error from the initial position to the first position on the reference path

However as the goal is to reach the end position $\boldsymbol{p}_d^e(10,\ \ 4)^T$ for this example, this is not the optimal way to follow this path. This is because we would have to travel backwards along the path before following the path, and thus the distance traveled whilst not on the path would increase. We will instead argue that it is better to converge to the path as quickly as possible, before following the path to the end point. The motivation behind this is that the main goal is to follow the path to the end point, and therefore any past path positions, i.e path positions further away from the end point than the vehicle, can be disregarded. By instead using the parameter s as an optimization variable, one would be able to not only choose the optimal input to reach the path, but also the optimal position on the path to reach. As it is desirable to reach the path as soon as possible regardless of initial position, the reference position should thus be chosen such that it minimizes the error from the initial position towards the path. By using Pythagoras' theorem, the distance e from the initial position $\boldsymbol{p}^e(0)$ to a reference position $\boldsymbol{p}_d^e$ can be written as

$$e^2 = (x_d^e - x^e(0))^2 + (y_d^e - y^e(0))^2 \tag{4.5}$$

Our goal is to minimize $|e|$, as this will yield the reference position that is closest to the initial position. As $min\,|e| \iff min\,e^2$ for the scalar value e, we can introduce a minimization problem

$$min\ \ (\boldsymbol{p}_d^e - \boldsymbol{p}^e(0))^T \boldsymbol{Q} (\boldsymbol{p}_d^e - \boldsymbol{p}(0)) \tag{4.6}$$

By choosing $\boldsymbol{Q}$ as a two by two identity matrix, the objective function in (4.6) will be equal to the right hand side of (4.5). Thus the minimum total error from the reference, and by extension optimal $\boldsymbol{p}_d^e$ is found by solving this minimization problem. The weighting matrix $\boldsymbol{Q}$ is set as the identity matrix for this problem, as there is no additional cost for neither x_d^e nor y_d^e, however this could be modified if we wished to penalize one error more than the other. Further, all solutions to this problem must be constrained such that

$\boldsymbol{p}_d^e = \Lambda(s)$, as this will ensure that any reference positions is on the path $\boldsymbol{\Lambda}$. The optimal reference position is thus found by solving

$$min \quad \begin{bmatrix} x^e - x^e(0) & y_d^e - y^e(0) \end{bmatrix} \begin{bmatrix} x_d^e - x^e(0) \\ y_d^e - y^e(0) \end{bmatrix} \tag{4.7}$$

subject to

$$x_d^e = \lambda_x(s) \tag{4.8a}$$

$$y_d^e = \lambda_y(s) \tag{4.8b}$$

$$0 \leq s \leq n \tag{4.8c}$$

By inserting the values from the example in (4.4), we get

$$min \quad \begin{bmatrix} x^e - 2 & y_d^e - 3 \end{bmatrix} \begin{bmatrix} x_d^e - 2 \\ y_d^e - 3 \end{bmatrix} \tag{4.9}$$

subject to

$$x_d^e = 0.5s \tag{4.10a}$$

$$y_d^e = 0.2s \tag{4.10b}$$

$$0 \leq s \leq 20 \tag{4.10c}$$

and the optimal solution to this is found as

$$\boldsymbol{p}_d^e = \begin{bmatrix} x_d^e \\ y_d^e \end{bmatrix} = \begin{bmatrix} 2.7586 \\ 1.1034 \end{bmatrix} \tag{4.11}$$

This is illustrated in **Fig. 4.2**, and for comparison the total error from the path is $e = 2.04$ in **Fig. 4.2** whilst it is $e = 3.61$ in **Fig. 4.1**.

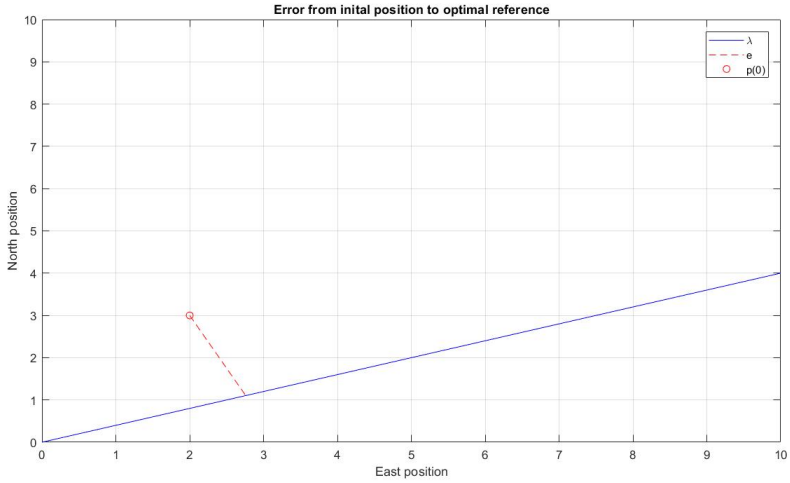

Figure 4.2: Error from the initial position to the optimal position on the reference path

As we have now derived a method of choosing the optimal path position to reach, the next goal should be to derive a control law that ensures convergence to the path before following the path to the end point. One method for reaching a reference with an MPC, is by changing the MPC formulation from Foss and Heirung (2016). As the objective function given in (2.9) aims to drive the system states to zero, this must be modified in order to converge the states towards a reference value r. This is achieved by ensuring that $r - x = 0$. The objective function can thus be modified to minimize $|r - x|$ instead of $|x|$, as the minimal value of $|r - x| = 0$. We can thus reformulate the objective function as follows

$$min \sum_{k=1}^{k=N} (r(k) - x(k))^T Q(r(k) - x(k)) + u(k)^T R u(k) \tag{4.12}$$

We also note that (2.6) and (4.12) are equal for $r = 0$. By introducing the ENU-position and the desired ENU-position as states and references the optimization function can be written as

$$min \;\; (\boldsymbol{P}_d^e - \boldsymbol{P}^e)^T \boldsymbol{Q} (\boldsymbol{P}_d^e - \boldsymbol{P}^e) + \boldsymbol{\mathcal{T}}^T \boldsymbol{R} \boldsymbol{\mathcal{T}} \tag{4.13}$$

subject to

$$\boldsymbol{A}_{eq} \begin{bmatrix} \boldsymbol{P}^e \\ \boldsymbol{\mathcal{T}} \end{bmatrix} = \boldsymbol{B}_{eq} \tag{4.14}$$

where $\boldsymbol{A}_{eq}$ and $\boldsymbol{B}_{eq}$ represents the system dynamics as in (2.5). We note here that for our system (2.9) and (4.12) are equal if the reference value $r(k) = 0$. We can notice that the objective function similar to the objective function for finding the optimal path position as described in (4.9), however we now also include the full prediction horizon for the MPC as well as a cost for the inputs. Thus, by expanding the equality constraints in (4.14) to also include the constraints given in (4.8), a controller can be derived that will find the optimal path position as well as the optimal input to reach this path position.

For this algorithm, we will have four optimization variables, $\boldsymbol{p}_d^e$, s, $\boldsymbol{p}^e$ and $\boldsymbol{\tau}$, which will be optimized for the entire prediction horizon. This optimization function will be constrained by the system dynamics, as well as $\boldsymbol{p}_d^e = \Lambda(s)$. We thus get the objective function

$$min \;\; \begin{bmatrix} S^T & (\boldsymbol{P}_d^e)T & (\boldsymbol{P}^e)^T \end{bmatrix} \boldsymbol{Q} \begin{bmatrix} S \\ \boldsymbol{P}_d^e \\ \boldsymbol{P}^e \end{bmatrix} + \boldsymbol{\mathcal{T}}^T \boldsymbol{R} \boldsymbol{\mathcal{T}} \tag{4.15}$$

where $\boldsymbol{Q}$ must be chosen such that we for every time step k have $(\boldsymbol{p}_d^e(k) - \boldsymbol{p}^e(k))^T Q (\boldsymbol{p}_d^e(k) - \boldsymbol{p}^e(k))$. Further, there will be no cost for S as s can be chosen freely to minimize the distance from the path at every time step. We will first examine the cost function where the prediction horizon is one. Without regarding the input, the cost function can for this

system be written as

$$\begin{bmatrix} s & x_d^e(1) & y_d^e(1) & x^e(1) & y^e(1) \end{bmatrix} \begin{bmatrix} q_{11} & q_{12} & q_{13} & q_{14} & q_{15} \\ q_{21} & q_{22} & q_{23} & q_{24} & q_{25} \\ q_{31} & q_{32} & q_{33} & q_{34} & q_{35} \\ q_{41} & q_{42} & q_{43} & q_{44} & q_{45} \\ q_{51} & q_{52} & q_{53} & q_{54} & q_{55} \end{bmatrix} \begin{bmatrix} s \\ x_d^e(1) \\ y_d^e(1) \\ x^e(1) \\ y^e(1) \end{bmatrix}$$
$$= q_2(x_d^e(1) - x^e(1))^2 + q_3(y_d^e(1) - y^e(1))^2 \tag{4.16}$$

which can be realized with

$$q_{22} = q_{42} = q_2 \tag{4.17a}$$

$$q_{24} = q_{44} = -q_2 \tag{4.17b}$$

$$q_{33} = q_{53} = q_3 \tag{4.17c}$$

$$q_{35} = q_{55} = -q_3 \tag{4.17d}$$

and where the remaining elements are zero. As we wish to penalize error in x and y direction equally, we choose $q_2 = q_3 = q$. Expanding (4.16) to include the complete prediction horizon from $k = 1 \rightarrow N$, i.e with states

$$S = \begin{bmatrix} s(1) \\ s(2) \\ \vdots \\ s(N) \end{bmatrix} \tag{4.18a}$$

$$\boldsymbol{P}_d^e \begin{bmatrix} \boldsymbol{p}_d^e(1) \\ \boldsymbol{p}_d^e(2) \\ \vdots \\ \boldsymbol{p}_d^e(N) \end{bmatrix} \tag{4.18b}$$

$$\boldsymbol{P}^e \begin{bmatrix} \boldsymbol{p}^e(1) \\ \boldsymbol{p}^e(2) \\ \vdots \\ \boldsymbol{p}^e(N) \end{bmatrix} \tag{4.18c}$$

we get the following cost functions for each state

$$\boldsymbol{Q_s} = \begin{bmatrix} 0 & \dots & \dots & 0 \\ \vdots & \ddots & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \dots & \dots & 0 \end{bmatrix} \tag{4.19a}$$

$$\boldsymbol{Q_d} = \begin{bmatrix} q\boldsymbol{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & q\boldsymbol{I} \end{bmatrix} \tag{4.19b}$$

$$\boldsymbol{Q_p} = \begin{bmatrix} -q\boldsymbol{I} & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \mathbf{0} \\ \mathbf{0} & \dots & \mathbf{0} & -q\boldsymbol{I} \end{bmatrix} \tag{4.19c}$$

where $\boldsymbol{I}$ is a 2×2 identity matrix and $\mathbf{0}$ is a 2×2 matrix of zeros. The complete cost matrix $\boldsymbol{Q}$ for this control law can thus be written as

$$\boldsymbol{Q} = \begin{bmatrix} \boldsymbol{Q_s} & \mathbf{0}_{N\times 2N} & \mathbf{0}_{N\times 2N} \\ \mathbf{0}_{2N\times 2N} & \boldsymbol{Q_d} & \boldsymbol{Q_p} \\ \mathbf{0}_{2N\times 2N} & \boldsymbol{Q_d} & \boldsymbol{Q_p} \end{bmatrix} \tag{4.20}$$

and we thus have the objective function

$$min \quad \begin{bmatrix} S^T & (\boldsymbol{P}_d^e)^T & (\boldsymbol{P}^e)^T(\boldsymbol{\mathcal{T}})^T \end{bmatrix} \begin{bmatrix} \boldsymbol{Q} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} S \\ \boldsymbol{P}_d^e \\ \boldsymbol{P}^e \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \tag{4.21}$$

subject to

$$\boldsymbol{A}_{eq} \begin{bmatrix} S \\ \boldsymbol{P}_d^e \\ \boldsymbol{P}^e \\ \boldsymbol{\mathcal{T}} \end{bmatrix} = \boldsymbol{B}_{eq} \tag{4.22}$$

By implementing this control law, we will get a controller that drives the system towards the optimal path position i.e the closest position on the path to the initial position, regardless of initial position. We have thus achieved convergence to the path, however this control law will not move any further along the path once it has reached the path. As we wish to follow the path towards the end point after convergence, this control law will have to be expanded.

We will now derive a new control law that will ensure that we follow the path to the end point when the path is reached, as well as convergence to the optimal point on the path. There are several different ways this can be done, however we will now propose a new method to achieve this. In the control law from (4.21) we are minimizing the error $(x_d^e(k) - x^e(k))^2 + (y_d^e(k) - y^e(k))^2$, which is the goal to reach any reference. However, we will now instead impose this as an additional equality constraint, i.e $\boldsymbol{P}^e = \boldsymbol{P}_d^e$. This constraint however will for most cases give infeasible solutions to the optimization, as there might not exist control inputs that are able to solve this for large initial errors. We will therefore introduce a vector of slack variables, $\boldsymbol{\epsilon} = [\epsilon_x \;\; \epsilon_y]^T$, such that $\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) = \boldsymbol{p}_d^e(k)$. As $\boldsymbol{\epsilon}$ can be chosen arbitrarily large, this constraint will always be satisfied, and we

can thus guarantee that there always will be a feasible solution to the optimization. Further, as our goal is to achieve $\boldsymbol{p}^e(k) = \boldsymbol{p}_d^e(k)$ for any k, this is equivalent with $\boldsymbol{\epsilon} \rightarrow [0 \;\; 0]^T$, which can be realized by including

$$min \;\; \boldsymbol{\epsilon}^T Q_\epsilon \boldsymbol{\epsilon} \tag{4.23}$$

where the cost matrix Q_ϵ will be chosen such that it ensures rapid convergence to zero. Further, as $\boldsymbol{p}_d^e = \Lambda(s)$, $\boldsymbol{P}_d^e$ can be removed from the optimization. By introducing

$$\boldsymbol{\mathcal{E}} = \begin{bmatrix} \boldsymbol{\epsilon}(1) \\ \boldsymbol{\epsilon}(2) \\ \vdots \\ \boldsymbol{\epsilon}(N) \end{bmatrix} \tag{4.24}$$

the optimization function from (4.21) and (4.22) can be rewritten as

$$min \;\; \begin{bmatrix} S^T & \boldsymbol{\mathcal{E}}^T & (\boldsymbol{P}^e)^T & \boldsymbol{\mathcal{T}}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{0} & \dots & \dots & \boldsymbol{0} \\ \boldsymbol{0} & \boldsymbol{Q}_\epsilon & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \ddots & \boldsymbol{0} & \boldsymbol{0} \\ \boldsymbol{0} & \dots & \boldsymbol{0} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} S^T \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{P}^e \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \tag{4.25}$$

subject to

$$\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) = \Lambda(s) \tag{4.26a}$$

$$\boldsymbol{p}^e(k) = A\boldsymbol{p}^e(k-1) + B\boldsymbol{\tau}(k-1) \tag{4.26b}$$

We note here that there is no cost matrix for $\boldsymbol{P}^e$, or rather, all the elements are zero, which implies that the value of $\boldsymbol{P}^e$ does not directly affect the function value of the objective function. The reason for this is that any additional cost on $\boldsymbol{P}^e$ would attempt to drive $\boldsymbol{p}^e(k)$ towards zero, which is not necessarily a position on the path. Further, by minimizing $\boldsymbol{\mathcal{E}}$, we ensure that $\boldsymbol{p}^e(k)$ converges to the path.

As this is just a reformulation of (4.21) and (4.22), we have still not ensured that we follow the path towards the endpoint. In order to reach the end point $\boldsymbol{p}_f^e = (x_f^e, \;\; y_f^e)^T$, we wish to minimize the error

$$e = (x^e - x_f^e)^2 + (y^e - y_f^e)^2 \tag{4.27}$$

However, as the position is already constrained to be on the path by (4.26a), we need only converge s towards the end point to achieve this objective. As $s = 0$ is defined as the end point for our path definition, the end point can be reached by implementing the minimization

$$min \;\; s^T Q_s s \tag{4.28}$$

This can further be expanded to include the entire prediction horizon

$$min \;\; S^T \boldsymbol{Q}_s S \tag{4.29}$$

where

$$\boldsymbol{Q}_e = \begin{bmatrix} q_s & 0 & \dots & 0 \\ 0 & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \dots & 0 & q_s \end{bmatrix} \tag{4.30}$$

Next we will discuss the choice of the weighting matrices Q_ϵ and Q_s. Both have an important function in the control law as Q_ϵ ensures that we converge the desired path whilst Q_s ensures that we converge towards the desired end point. As we wish to converge at an equal rate in both x and y direction, the Q_ϵ matrix will be given

$$Q_\epsilon = \begin{bmatrix} q_\epsilon & 0 \\ 0 & q_\epsilon \end{bmatrix} \tag{4.31}$$

and as s is a scalar parameter, Q_s will be

$$Q_e = q_s \tag{4.32}$$

Further, as we wish to converge to the end point along the path, we wish to impose a larger weight on $\boldsymbol{\epsilon}$ than s. This is will ensure more rapid convergence towards the path than towards the end point, which is equivalent of prioritizing to reach the path before following the path towards the end point. This can be realized by choosing

$$q_s \ll q_\epsilon \tag{4.33}$$

which will in theory ensure that we only converge towards the end point once we have reached the path. We can thus formulate the optimization problem as

$$min \quad \begin{bmatrix} S^T & \boldsymbol{\mathcal{E}}^T & (\boldsymbol{P}^e)^T & \boldsymbol{\mathcal{T}}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{Q_s} & \boldsymbol{0}_{N\times 2N} & \boldsymbol{0}_{N\times 2N} & \boldsymbol{0}_{N\times 2N} \\ \boldsymbol{0}_{2N\times N} & \boldsymbol{Q}_\epsilon & \boldsymbol{0}_{2N\times 2N} & \boldsymbol{0}_{2N\times 2N} \\ \boldsymbol{0}_{2N\times N} & \boldsymbol{0}_{2N\times 2N} & \boldsymbol{0}_{2N\times 2N} & \boldsymbol{0}_{2N\times 2N} \\ \boldsymbol{0}_{2N\times N} & \boldsymbol{0}_{2N\times 2N} & \boldsymbol{0}_{2N\times 2N} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{P}^e \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \tag{4.34}$$

subject to

$$\boldsymbol{P}^e + \boldsymbol{\mathcal{E}} = \Lambda(S) \tag{4.35a}$$

$$\boldsymbol{p}^e(1) - \boldsymbol{B\tau}(0) = \boldsymbol{A}\boldsymbol{p}^e(0) \tag{4.35b}$$

$$\boldsymbol{p}^e(k) - \boldsymbol{A}\boldsymbol{p}^e(k-1) - \boldsymbol{B\tau}(k-1) = \boldsymbol{0}, \quad k \in [2, \ \ N] \tag{4.35c}$$

This optimization function will serve as the basis for all control laws of this thesis. As this optimization function is derived for 2-DOF, i.e only the position $\boldsymbol{p}^e$ is optimized without regarding orientation, the states in (4.34) and (4.35) will need to be expanded to account for vehicle orientation.

4.3 Linearized MPC

The first control law we will derive based on the optimization from **Section 6.1** will be a linearizing MPC. For this approach we will attempt to implement a linear MPC for 3-DOF, by linearizing the UGV dynamics for each time step. The required states for this control law are the states given in the 3-DOF model in Dyrnes (2018), i.e $[x^e \ y^e \ \psi]^T$. Further the linear damping dynamics will be used for the yaw rate, i.e

$$\begin{bmatrix} \dot{\psi} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} 0 & 1 \\ 0 & -\frac{D}{I_z} \end{bmatrix} \begin{bmatrix} \psi \\ r \end{bmatrix} + \begin{bmatrix} 0 \\ \frac{1}{I_z} \end{bmatrix} T_z \tag{4.36}$$

with T_z being the total torque applied to the vehicle. Further the torque has been derived as

$$T_z = u|u|\mu_f R \sin\gamma \tag{4.37}$$

with u and γ being the body-fixed velocity and steering angle respectively. The nonlinear UGV model in 3-DOF is thus given

$$\dot{\boldsymbol{x}} = \begin{bmatrix} \dot{x}^e \\ \dot{y}^e \\ \dot{\psi} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} f_1(\boldsymbol{x}, \boldsymbol{\tau}) \\ f_2(\boldsymbol{x}, \boldsymbol{\tau}) \\ f_3(\boldsymbol{x}, \boldsymbol{\tau}) \\ f_4(\boldsymbol{x}, \boldsymbol{\tau}) \end{bmatrix} \begin{bmatrix} u\cos\psi \\ u\sin\psi \\ r \\ \frac{1}{I_z}(u|u|\mu_f R\sin\gamma - D\, r) \end{bmatrix} \tag{4.38}$$

where $\boldsymbol{x} = [x^e \ \ y^e \ \ \psi \ \ r]^T$ and $\boldsymbol{\tau} = [u \ \ \gamma]^T$. By linearization, the system is given in state space form as follows

$$\begin{aligned} \begin{bmatrix} \dot{x}^e \\ \dot{y}^e \\ \dot{\psi} \\ \dot{r} \end{bmatrix} &= \boldsymbol{J} \begin{bmatrix} x^e \\ y^e \\ \psi \\ r \end{bmatrix} + \boldsymbol{B} \begin{bmatrix} u \\ \gamma \end{bmatrix} = \begin{bmatrix} \frac{\delta f_1}{\delta x^e} & \frac{\delta f_1}{\delta y^e} & \frac{\delta f_1}{\delta \psi} & \frac{\delta f_1}{\delta r} \\ \frac{\delta f_2}{\delta x^e} & \frac{\delta f_2}{\delta y^e} & \frac{\delta f_2}{\delta \psi} & \frac{\delta f_2}{\delta r} \\ \frac{\delta f_3}{\delta x^e} & \frac{\delta f_3}{\delta y^e} & \frac{\delta f_3}{\delta \psi} & \frac{\delta f_3}{\delta r} \\ \frac{\delta f_4}{\delta x^e} & \frac{\delta f_4}{\delta y^e} & \frac{\delta f_4}{\delta \psi} & \frac{\delta f_4}{\delta r} \end{bmatrix} \begin{bmatrix} x^e \\ y^e \\ \psi \\ r \end{bmatrix} + \begin{bmatrix} \frac{\delta f_1}{\delta u} & \frac{\delta f_1}{\delta \gamma} \\ \frac{\delta f_2}{\delta u} & \frac{\delta f_2}{\delta \gamma} \\ \frac{\delta f_3}{\delta u} & \frac{\delta f_3}{\delta \gamma} \\ \frac{\delta f_4}{\delta u} & \frac{\delta f_4}{\delta \gamma} \end{bmatrix} \begin{bmatrix} u \\ \gamma \end{bmatrix} \\ &= \begin{bmatrix} 0 & 0 & -u^* \sin\psi^* & 0 \\ 0 & 0 & u^* \cos\psi^* & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -\frac{D}{I_z} \end{bmatrix} \begin{bmatrix} x^e \\ y^e \\ \psi \\ r \end{bmatrix} + \begin{bmatrix} \cos\psi^* & 0 \\ \sin\psi^* & 0 \\ 0 & 0 \\ \frac{2u^*\mu_f R\sin\gamma^*}{I_z} & \frac{u^*|u^*|\mu_f R\cos\gamma^*}{I_z} \end{bmatrix} \begin{bmatrix} u \\ \gamma \end{bmatrix} \end{aligned} \tag{4.39}$$

where $*$ indicates that the value for the point of linearization should be inserted, e.g if we linearize about $u = 0.5$ we have $u^* = 0.5$.

4.3.1 Controllability

Before moving further with the control law, controllability for the linearized system should be checked. Controllability of this linearized system can be shown by calculating the controllability matrix. For $J \in \mathcal{R}^{n\times n}$, we have the controllability matrix

$$\boldsymbol{\mathcal{C}} = \begin{bmatrix} \boldsymbol{B} & \boldsymbol{JB} & \boldsymbol{J}^2\boldsymbol{B} & \dots & \boldsymbol{J}^{n-1}\boldsymbol{B} \end{bmatrix} \tag{4.40}$$

which for the linearized sytem in 3-DOF yields the following controllability matrix

$$\mathcal{C} = \begin{bmatrix} \cos\psi & 0 & 0 & 0 \\ \sin\psi & 0 & 0 & 0 \\ 0 & 0 & \frac{2u\mu_f R\sin\gamma}{I_z} & \frac{u|u|\mu_f R\cos\gamma}{I_z} \\ \frac{2u\mu_f R\sin\gamma}{I_z} & \frac{u|u|\mu_f R\cos\gamma}{I_z} & \frac{-2Du\mu_f R\sin\gamma}{I_z^2} & \frac{-2Du|u|\mu_f R\cos\gamma}{I_z^2} \\ \frac{-2u^2\mu_f R\sin\psi\sin\gamma}{I_z} & \frac{-u^2|u|\mu_f R\sin\psi\cos\gamma}{I_z} & \frac{2Du^2\mu_f R\sin\psi\sin\gamma}{I_z^2} & \frac{2Du^2|u|\mu_f R\sin\psi\cos\gamma}{I_z^2} \\ \frac{2u^2\mu_f R\cos\psi\sin\gamma}{I_z} & \frac{u^2|u|\mu_f R\cos\psi\cos\gamma}{I_z} & \frac{-2Du^2\mu_f R\cos\psi\sin\gamma}{I_z^2} & \frac{-2Du^2|u|\mu_f R\cos\psi\cos\gamma}{I_z^2} \\ \frac{-2Du\mu_f R\sin\gamma}{I_z^2} & \frac{-2Du|u|\mu_f R\cos\gamma}{I_z^2} & \frac{2D^2u\mu_f R\sin\gamma}{I_z^3} & \frac{2D^2u|u|\mu_f R\cos\gamma}{I_z^3} \\ \frac{2D^2u\mu_f R\sin\gamma}{I_z^3} & \frac{2D^2u|u|\mu_f R\cos\gamma}{I_z^3} & \frac{-2D^3u\mu_f R\sin\gamma}{I_z^4} & \frac{-2D^3u|u|\mu_f R\cos\gamma}{I_z^4} \end{bmatrix} \tag{4.41}$$

The $*$ symbol is dropped to simplify the notation, however the values for the point of linearization should still be inserted. Controllability for the system can thus be checked by examining the rank of the controllability matrix. A system is controllable if the rank of the controllability matrix is equal to the number of states. This system will thus be controllable if we have $rank(\mathcal{C}) = 4$. We therefore check controllability by inserting values in the controllabillity matrix. The first thing we notice is that for the initial condition, i.e $u = 0$, we have

$$\mathcal{C} = \begin{bmatrix} \cos\psi & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ \sin\psi & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.42}$$

The rows and columns containing only zeros can be cut from the matrix as they do not add to the rank, which gives

$$\mathcal{C} = \begin{bmatrix} \cos\psi \\ \sin\psi \end{bmatrix} \tag{4.43}$$

and $rank(\mathcal{C}) = 1$, as we only have one column. This implies that the system is not controllable for the initial velocity input $u = 0$. Further by choosing any $u \neq 0$ and inserting a value of $\gamma = 0$, we get

$$\begin{aligned} \mathcal{C} &= \begin{bmatrix} \cos\psi & 0 & 0 & 0 & 0 & \frac{-u^2|u|\mu_f R\sin\psi}{I_z} & 0 & \frac{2Du^2|u|\mu_f R\sin\psi}{I_z^2} \\ \sin\psi & 0 & 0 & 0 & 0 & \frac{u^2|u|\mu_f R\cos\psi}{I_z} & 0 & \frac{-2Du^2|u|\mu_f R\cos\psi}{I_z^2} \\ 0 & 0 & 0 & \frac{u|u|\mu_f R}{I_z} & 0 & \frac{-2Du|u|\mu_f R}{I_z^2} & 0 & \frac{2D^2u|u|\mu_f R}{I_z^3} \\ 0 & \frac{u|u|\mu_f R}{I_z} & 0 & \frac{-2Du|u|\mu_f R}{I_z^2} & 0 & \frac{2D^2u|u|\mu_f R}{I_z^3} & 0 & \frac{-2D^3u|u|\mu_f R}{I_z^4} \end{bmatrix} \\ &= \begin{bmatrix} \cos\psi & 0 & 0 & \frac{-u^2|u|\mu_f R\sin\psi}{I_z} & \frac{2Du^2|u|\mu_f R\sin\psi}{I_z^2} \\ \sin\psi & 0 & 0 & \frac{u^2|u|\mu_f R\cos\psi}{I_z} & \frac{-2Du^2|u|\mu_f R\cos\psi}{I_z^2} \\ 0 & 0 & \frac{u|u|\mu_f R}{I_z} & \frac{-2Du|u|\mu_f R}{I_z^2} & \frac{2D^2u|u|\mu_f R}{I_z^3} \\ 0 & \frac{u|u|\mu_f R}{I_z} & \frac{-2Du|u|\mu_f R}{I_z^2} & \frac{2D^2u|u|\mu_f R}{I_z^3} & \frac{-2D^3u|u|\mu_f R}{I_z^4} \end{bmatrix} \end{aligned} \tag{4.44}$$

For this case, we will have $rank(\boldsymbol{\mathcal{C}}) = 4$ if we have four linearly independent rows and columns. According to Miller (2011), two rows or columns are said to be linearly dependent if there exist two scalars $x_1 \neq 0$ and $x_2 \neq 0$ such that

$$\boldsymbol{\mathcal{C}}_i x_1 + \boldsymbol{\mathcal{C}}_j x_2 = 0 \tag{4.45}$$

or equivalently if there exists one scalar $a \neq 0$ such that $\boldsymbol{\mathcal{C}}_i - a\boldsymbol{\mathcal{C}}_j = \mathbf{0}$, where i and j are the row or column numbers and $\mathbf{0}$ is a vector of zeros. The parameters D, I_z and μ_f will be strictly positive scalars by definition, as any other values would imply zero or negative mass or friction. First of all we can easily verify that we have at least four linearly independent columns, as the first three columns have values and zeros in different rows, e.g the first column has values in row one and two whilst column two only have a value in row four. As both column four and five have non zero elements in all four rows, they are both linearly independent from the first three columns as the first three columns all contain zero elements. This implies that we have at least four linearly independent columns. By using the same method, we can also see that both two bottom rows are linearly independent of each other as row four has a non zero element in the second column whilst the third row has a zero element in the third column. Similarly we can verify that both bottom rows are linearly independent of the top two rows, as both bottom rows has a non zero element in the fourth column. Further, when comparing the top two rows, we can see that

$$\begin{bmatrix} \cos\psi & \frac{-u^2|u|\mu_f R\sin\psi}{I_z} & \frac{2Du^2|u|\mu_f R\sin\psi}{I_z^2} \end{bmatrix} \neq a \begin{bmatrix} \sin\psi & \frac{u^2|u|\mu_f R\cos\psi}{I_z} & \frac{-2Du^2|u|\mu_f R\cos\psi}{I_z^2} \end{bmatrix}, \quad \forall \quad \psi, \ a \tag{4.46}$$

which implies that the top two rows are linearly independent, and that we have four linearly independent rows. As we have both four linearly independent rows and columns, we have $rank(\boldsymbol{\mathcal{C}}) = 4$, which shows controllabillity for $\gamma = 0$. Controllability should now be checked for other values of γ, however we first note some properties of γ. As the vehicle has a physical constraint on the steering angle, i.e $-35^\circ \leq \gamma \leq 35^\circ$, we therefore have $\cos\gamma > 0$ and $\cos\gamma > \sin\gamma$. These properties may simplify the controllabillity proof, as no rows or columns will be made zero caused by $\gamma = 90^\circ$. Nor will any rows or columns be linearly dependant as $\cos\gamma \neq \sin\gamma$. Further, controllability could be shown by using the full matrix in (4.41), however because of the aforementioned properties, it might be sufficient to use the simplified matrix from (4.44). As mentioned earlier, we need at least four linearly independent rows and columns, and might therefore not be necessary to calculate for the entire controllability matrix from (4.41). By inserting γ in (4.44), we get

$$\boldsymbol{\mathcal{C}} = \begin{bmatrix} \cos\psi & 0 & 0 & \frac{-u^2|u|\mu_f R\sin\psi\cos\gamma}{I_z} & \frac{2Du^2|u|\mu_f R\sin\psi\cos\gamma}{I_z^2} \\ \sin\psi & 0 & 0 & \frac{u^2|u|\mu_f R\cos\psi\cos\gamma}{I_z} & \frac{-2Du^2|u|\mu_f R\cos\psi\cos\gamma}{I_z^2} \\ 0 & 0 & \frac{u|u|\mu_f R\cos\gamma}{I_z} & \frac{-2Du|u|\mu_f R\cos\gamma}{I_z^2} & \frac{2D^2u|u|\mu_f R\cos\gamma}{I_z^3} \\ \frac{2u\mu_f R\sin\gamma}{I_z} & \frac{u|u|\mu_f R\cos\gamma}{I_z} & \frac{-2Du|u|\mu_f R\cos\gamma}{I_z^2} & \frac{2D^2u|u|\mu_f R\cos\gamma}{I_z^3} & \frac{-2D^3u|u|\mu_f R\cos\gamma}{I_z^4} \end{bmatrix} \tag{4.47}$$

As $\cos\gamma \neq 0$, we have at least four linearly independent columns as for (4.44). Further, the argument for linear independence for the bottom rows still holds, as none of the elements are made zero for any valid value of γ. We also know that as the two top rows

where linearly independent for $\cos\gamma = 1$, inserting any new value of γ would be equal to multiplying by a factor of $\cos\gamma$. As the definition of linear independence states that there is no value of a that will make the two rows equal, there can not exist a value of $\cos\gamma$ that will make the two rows equal as this would be equivalent of multiplying with a constant. Therefore the top two rows must still be linearly independent for any valid value of γ, and we therefore have

$$rank(\boldsymbol{\mathcal{C}}) = 4, \qquad \forall \;\; u \neq 0 \;\; q.e.d \tag{4.48}$$

Thus the linearized system is controllable for all values except for $u = 0$. This does however pose one problem, as $u = 0$ is a valid input for the control problem, and in fact will be the initial condition for most applications. There is however a solution to this problem. If we choose to never linearize about $u = 0$, i.e selecting another value of u for the point of linearization, this problem should be avoided. Even though this will bring more inaccuracy to the controller, as a linearized model only is accurate around the point of linearization, one might argue that for a sufficiently small u this linearization error will be negligible. We will therefore proceed with this control algorithm by linearizing about a small value of u when we in reality have $u = 0$, and this value will be tuned by testing later.

4.3.2 Control law

Before implementing the system dynamics as equality constraints, the state space model can further be discretized by the use of the first order explicit Euler discretization method, i.e

$$X(k+1) = \boldsymbol{A}_d(k)X(k) + \boldsymbol{B}_d(k)\boldsymbol{\tau}(k) \tag{4.49}$$

with

$$A_d = I + hJ = \begin{bmatrix} 1 & 0 & -h\,u^* \sin\psi^* & 0 \\ 0 & 1 & h\,u^* \cos\psi^* & 0 \\ 0 & 0 & 1 & 1 \\ 0 & 0 & 0 & 1 - \frac{h\,D}{I_z} \end{bmatrix} \tag{4.50}$$

$$B_d = hB = \begin{bmatrix} h\cos\psi^* & 0 \\ h\sin\psi^* & 0 \\ 0 & 0 \\ \frac{2u^* h\,\mu_f R \sin\gamma^*}{I_z} & \frac{u^*|u^*|h\,\mu_f R\cos\gamma^*}{I_z} \end{bmatrix} \tag{4.51}$$

where h is the sampling time. As the linear discrete state space model describes the change of $\boldsymbol{x}$, it will serve as the linear equality constraints for the system for every time step similar to what is done in Foss and Heirung (2016), and by using (2.3), we can express the system on the form

$$\begin{aligned} \boldsymbol{x}(1) &= \boldsymbol{A}_d(0)\boldsymbol{x}(0) + \boldsymbol{B}_d(0)\boldsymbol{\tau}(0) \\ \boldsymbol{x}(2) &= \boldsymbol{A}_d(1)\boldsymbol{x}(1) + \boldsymbol{B}_d(1)\boldsymbol{\tau}(1) \\ &\vdots \\ \boldsymbol{x}(N) &= \boldsymbol{A}_d(N-1)\boldsymbol{x}(N-1) + \boldsymbol{B}_d(N-1)\boldsymbol{\tau}(N-1) \end{aligned} \tag{4.52}$$

Further, by introducing the variable

$$\boldsymbol{X} = \begin{bmatrix} \boldsymbol{x}(1) \\ \boldsymbol{x}(2) \\ \vdots \\ \boldsymbol{x}(N) \end{bmatrix} \tag{4.53}$$

(4.52) can thus be rearranged to matrix form similar to (2.4) as

$$\underbrace{\begin{bmatrix} \boldsymbol{I}_{4\times4} & \boldsymbol{0}_{4\times4} & \dots & \dots & \boldsymbol{0}_{4\times4} & -\boldsymbol{B}_d & \boldsymbol{0}_{4\times2} & \dots & \dots & \boldsymbol{0}_{4\times2} \\ -\boldsymbol{A}_d & \ddots & \ddots & & \vdots & \boldsymbol{0}_{4\times2} & \ddots & \ddots & & \vdots \\ \boldsymbol{0}_{4\times4} & \ddots & \ddots & \ddots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \ddots & \boldsymbol{0}_{4\times4} & \vdots & & \ddots & \ddots & \boldsymbol{0}_{4\times2} \\ \boldsymbol{0}_{4\times4} & \dots & \boldsymbol{0}_{4\times4} & -\boldsymbol{A}_d & \boldsymbol{I} & \boldsymbol{0}_{4\times2} & \dots & \dots & \boldsymbol{0}_{4\times2} & -\boldsymbol{B}_d \end{bmatrix}}_{\boldsymbol{A}_{eqd}} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix} = \underbrace{\begin{bmatrix} \boldsymbol{A}_d\boldsymbol{x}(0) \\ \boldsymbol{0}_{4\times1} \\ \vdots \\ \vdots \\ \boldsymbol{0}_{4\times1} \end{bmatrix}}_{\boldsymbol{B}_{eqd}} \tag{4.54}$$

which gives the equality constraints for the system dynamics

$$\boldsymbol{A}_{eqd} \begin{bmatrix} \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix} = \boldsymbol{B}_{eqd} \tag{4.55}$$

where $\boldsymbol{X}$ and $\boldsymbol{\mathcal{T}}$ are the states and inputs for the entire prediction horizon respectively. Further, we wish to implement the path constraints as described in (4.35a), which can be sorted to

$$\boldsymbol{\mathcal{E}} + \boldsymbol{P}^e - \Lambda(S) = \boldsymbol{0}_{2N\times1} \tag{4.56}$$

At this point it is important to note that in order to implement these constraints for the linear MPC, we require linear equality constraints, and by extension a linear path $\boldsymbol{\Lambda}$, such that

$$\boldsymbol{\Lambda} = \begin{bmatrix} \lambda_x \\ \lambda_y \end{bmatrix} s, \qquad s \in [0, \ \ n] \tag{4.57}$$

where λ_x and λ_y are constants. However, for this part we will assume that this holds and that we have a linear path when using this controller. Thus we will choose the path $\boldsymbol{\Lambda}_1$ given in (4.2), such that the path can be implemented in the linear equality constraints for the path, and (4.56) can be written on matrix for as

$$\boldsymbol{A}_{eqp} \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{X} \end{bmatrix} = \boldsymbol{B}_{eqp} \tag{4.58}$$

with

$$\boldsymbol{A}_{eqp} = \begin{bmatrix} \Lambda & \boldsymbol{0}_{2\times1} & \dots & \boldsymbol{0}_{2\times1} & \boldsymbol{I} & \boldsymbol{0}_{2\times2} & \dots & \boldsymbol{0}_{2\times2} & \boldsymbol{IO} & \boldsymbol{0}_{2\times4} & \dots & \boldsymbol{0}_{2\times4} \\ \boldsymbol{0}_{2\times1} & \ddots & \ddots & \vdots & \boldsymbol{0}_{2\times2} & \ddots & \ddots & \vdots & \boldsymbol{0}_{2\times4} & \ddots & \ddots & \vdots \\ \vdots & \ddots & \ddots & \boldsymbol{0}_{2\times1} & \vdots & \ddots & \ddots & \boldsymbol{0}_{2\times2} & \vdots & \ddots & \ddots & \boldsymbol{0}_{2\times4} \\ \boldsymbol{0}_{2\times1} & \dots & \boldsymbol{0}_{2\times1} & \Lambda & \boldsymbol{0}_{2\times2} & \dots & \boldsymbol{0}_{2\times2} & \boldsymbol{I} & \boldsymbol{0}_{2\times4} & \dots & \boldsymbol{0}_{2\times4} & \boldsymbol{IO} \end{bmatrix} \tag{4.59}$$

and

$$\boldsymbol{B}_{eqp} = \begin{bmatrix} \mathbf{0}_{2\times 1} \\ \vdots \\ \vdots \\ \mathbf{0}_{2\times 1} \end{bmatrix} \tag{4.60}$$

where $\boldsymbol{I}$ is a 2×2 identity matrix and

$$\boldsymbol{IO} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{4.61}$$

The full set of constraints for this system can thus be written as

$$\underbrace{\begin{bmatrix} \boldsymbol{A}_{eqp} & \mathbf{0}_{2N\times 2N} \\ \mathbf{0}_{4N\times 3N} & \boldsymbol{A}_{eqd} \end{bmatrix}}_{\boldsymbol{A}_{eq}} \underbrace{\begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix}}_{\boldsymbol{Z}} = \underbrace{\begin{bmatrix} \boldsymbol{B}_{eqp} \\ \boldsymbol{B}_{eqd} \end{bmatrix}}_{\boldsymbol{B}_{eq}} \tag{4.62}$$

We will now find the inequality constraints for the controller. As the system is bound by physical constraints on the input, these constraints should be enforced in the MPC to avoid inputs that can not be implemented. These physical constraints are the UGV's minimum and maximum velocity, as well as the minimum and maximum steering angle. These are $0\mathrm{m/s}$ and $5\mathrm{m/s}$, and -35° and 35° respectively. In reality, the UGV can have a negative velocity as well, i.e driving in reverse, however FFI does not desire this for the path following controllers.

Further, as a failsafe, FFI have imposed a maximum error for the vehicle. This is done by stopping the vehicle if the error, i.e deviation from the path, is greater than a predefined threshold. This can be implemented in the MPC by including a minimum and maximum value for the slack variables $\boldsymbol{\epsilon}$.

For the linearized MPC, we thus have the following inequality constraints

$$\boldsymbol{\tau}_{min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max} \tag{4.63a}$$

$$\boldsymbol{\epsilon}_{min} \leq \boldsymbol{\epsilon} \leq \boldsymbol{\epsilon}_{max} \tag{4.63b}$$

Thus, by inserting the states in the basic optimization from (4.34), we get the objective function

$$\begin{aligned} &min \;\; \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z} \\ &min \;\; \begin{bmatrix} S^T & \boldsymbol{\mathcal{E}}^T & (\boldsymbol{X})^T & \boldsymbol{\mathcal{T}}^T \end{bmatrix} \begin{bmatrix} \boldsymbol{Q}_s & \mathbf{0}_{N\times 2N} & \mathbf{0}_{N\times 4N} & \mathbf{0}_{N\times 2N} \\ \mathbf{0}_{2N\times N} & \boldsymbol{Q}_\epsilon & \mathbf{0}_{2N\times 4N} & \mathbf{0}_{2N\times 2N} \\ \mathbf{0}_{4N\times N} & \mathbf{0}_{4N\times 2N} & \mathbf{0}_{4N\times 4N} & \mathbf{0}_{4N\times 2N} \\ \mathbf{0}_{2N\times 2N} & \mathbf{0}_{2N\times 2N} & \mathbf{0}_{2N\times 4N} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \end{aligned} \tag{4.64}$$

As we for this controller for the linearized system have derived a quadratic objective function, as well as linear equality and inequality constraints, this problem can be solved as a

quadratic problem (QP). This gives the advantage that QP-solvers, which are proven to be efficient, can be implemented, which might give a reduced calculation time compared to a more complex nonlinear problem (NLP).

4.4 Nonlinear MPC for 3-DOF

The next controller we will attempt to implement is a NMPC for 3-DOF. The motivation behind this controller, is that the nonlinear dynamics will offer more accuracy compared to the linearized dynamics from **Section 4.3**, whilst less complex than a model for 6-DOF. As opposed to **Section 4.3**, this controller will be based on the the nonlinear damping model for the yaw dynamics. Further, instead of linearizing the dynamics for the operating point, the optimization will be solved using nonlinear solvers. We thus have the nonlinear model

$$\begin{bmatrix} \dot{x}^e \\ \dot{y}^e \\ \dot{\psi} \\ \dot{r} \end{bmatrix} = \begin{bmatrix} u\cos\psi \\ u\sin\psi \\ r \\ \frac{u|u|\mu_f R\sin\gamma}{I_z} - \frac{D}{I_z}\, r|r| \end{bmatrix} \tag{4.65}$$

where $[x^e \;\; y^e \;\; \psi \;\; \dot{\psi}]^T$ for the entire prediction horizon are the states $\boldsymbol{X}$, and $[u \;\; \gamma]^T$ are the control inputs $\boldsymbol{\tau}$. The linear damping model for the yaw rate dynamics is used for this control law, as testing of the model does not seem to offer any significant advantages to the nonlinear damping model in terms of accuracy. Therefore the linear damping model will be chosen, as this will give less computational complexity.

As we are still using the quadratic optimization function from **Section 4.2**, we can implement a similar control law as for **Section 4.3**, i.e

$$min \quad \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix}^T \begin{bmatrix} \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_\epsilon & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{Q}_e & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{R} \end{bmatrix} \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{X} \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \tag{4.66}$$

This time however, as we will use the nonlinear vehicle model, the equality constraints from (4.62) must be changed. By using Euler discretization, the UGV model from (4.65) can be written as

$$x(k) = x(k-1) + h\, u(k-1)\cos\psi(k-1) \tag{4.67a}$$

$$y(k) = y(k-1) + h\, u(k-1)\sin\psi(k-1) \tag{4.67b}$$

$$\psi(k) = \psi(k-1) + h\, r \tag{4.67c}$$

$$r(k) = r(k-1) + h\,(\frac{u(k-1)|u(k-1)|\mu_f R\sin\gamma(k-1)}{I_z} - \frac{D}{I_z} r(k-1)|r(k-1)|) \tag{4.67d}$$

Next, we will express the path constraints. As opposed to **Section 4.3**, we no longer require the path $\boldsymbol{\Lambda}$ to be a linear function s as we are no longer limited to linear constraints.

Therefore, this NMPC control law should be able to follow any path described by a non-linear function $\Lambda(s)$. The functions will vary for the different paths, however the general constraints will still be

$$\epsilon_x(k) + e_x(k) - \lambda_x(s(k)) = 0 \tag{4.68a}$$

$$\epsilon_y(k) + e_y(k) - \lambda_y(s(k)) = 0 \tag{4.68b}$$

As for the inequality constraints for this control law, we will still have the same constraints on the input as for **Section 4.3**, however we will now also impose a constraint on the heading angle ψ. The reason this is imposed is that as ψ is periodic, i.e $\psi = \psi + n2\pi \ \ \forall\, n \in \mathbb{N}$, any solution to the control law will have an equivalent solution with $\psi \in [0, \ \ 2\pi\rangle$, and by constraining ψ, we will limit the search for an optimal solution to within the bounds and thus we might reduce the computation time. Further, the measured angle will always be between 0 and 2π. However, as there is no physical constraint on the rotation of the vehicle, it is desirable for the controller to choose the shortest rotation to reach the desired ψ_d, e.g if we have $\psi = \frac{\pi}{4}$ and $\psi_d = \frac{7\pi}{4}$, the shortest rotation would be to rotate clockwise towards $\psi = -\frac{\pi}{4}$. Thus, to ensure that we consider the optimal rotation direction to reach ψ_d, the bound on ψ must be increased. By increasing the bound to $\psi \in [0, \ \ 4\pi\rangle$, we will consider counter clockwise rotation for $\psi_d < \psi - \pi$. Likewise, by expanding the lower bound to -2π, we will consider clockwise rotation for $\psi_d > \psi + \pi$. We thus have $\psi \in \langle -2\pi, \ \ 4\pi\rangle$

$$\boldsymbol{\tau}_{min} \leq \boldsymbol{\tau} \leq \boldsymbol{\tau}_{max} \tag{4.69}$$

4.5 Simplified Nonlinear MPC for 6-DOF

We will now derive a control law for the UGV in 6-DOF. The basics for this control law will be the same as for **Section 4.3** and **Section 4.4**, however this will be expanded for the entire 3D space. The motivation behind this is that by meassuring the UGV's orientation in the terrain and accounting for this in the control law, one might improve the accuracy as by accounting for how the UGV will move in the terrain.

We will however make one simplification for this control law. As the movement is bounded such that the UGV can not move independent from the ground, the pitch and roll angles θ and ϕ can not be controlled directly. These angles will solely depend on the inclination of the ground, and are thus not included in the objective function. However, as these angles affect the dynamics of the system, they will need to be included in the constraints. We will therefore derive a control law which uses linearizations of pitch and roll angles to describe the dynamics, whilst the nonlinear model is used for the remaining states. The motivation behind this is that this will reduce the complexity of the control law as we reduce the number of variables and nonlinear constraints.

As the MPC is robust with regards to modelling errors, and as only the first input in the input sequence is applied before the model is updated, we will attempt to simplify the model. By examining the kinematics given in (3.2) we notice that $\dot{x}^e$ and $\dot{y}^e$ are proportional to $\cos\theta$. If we consider θ to be constant for the entire prediction horizon at each sampling time, $\cos\theta$ will reduce to a constant c_θ in the range $c_\theta \in [0, \ \ 1]$. We note here that $\cos\theta = 0$ would give an uncontrollable system as this would give $\dot{x}^e = \dot{y}^e = 0$, however this would require the vehicle to be oriented vertically, which is outside of any

application of the UGV. The same can be done by examining the kinematics for $\dot{\psi}$, which is given

$$\dot{\psi} = q\frac{\sin\phi}{\cos\theta} + r\frac{\cos\phi}{\cos\theta} \tag{4.70}$$

If we for this control law consider any the roll and pitch angles as constant for each sampling time, and by extension the roll and pitch motion p and q as disturbances, the yaw velocity will be given

$$\dot{\psi} = r\frac{c_\phi}{c_\theta} \tag{4.71}$$

We note here that the singularity caused by $c_\theta = 0$ will not affect this system, as the pitch angle will never reach 90° as stated earlier.

By reducing the angles that can not be controlled directly, i.e θ and ϕ, to constants updated at each sampling time, we have reduced the complexity of the kinematics for 6-DOF. However as the effect these angles have on the UGV's motion at the current sampling time is still included, we have improved the accuracy from the 3-DOF model from **Section 4.4**. We thus have increased the accuracy of the model, without significantly increasing the computational complexity, which might improve the performance of this control law compared to the control law from **Section 4.4**.

One important thing to note for this controller is that there will be physical constraints on the pitch and roll angles, which will affect the validity of the dynamic model. These physical constraints are caused by the maximum inclination the vehicle will be able to handle. As a large pitch angle would cause the UGV's CG to shift past the contact point with the ground, i.e the CG is behind the rear wheels, this would cause the front wheels to lose contact with the ground and thus the vehicle would overturn. This is illustrated in **Fig. 4.3**, and as the model requires that all wheels are in contact with the ground, these constraints should not be violated.

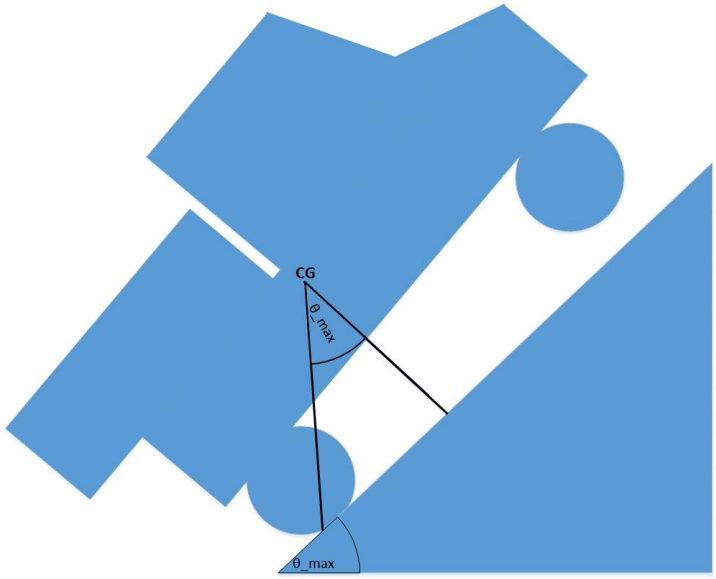

Figure 4.3: Illustration of vehicle overturning from Dyrnes (2018)

Dyrnes (2018) calculated the maximum pitch angle to be $\theta_{max} = 52.8°$ and the minimum pitch angle, i.e the steepest decline the UGV can handle was calculated to $\theta_{min} = -64.5°$. The same applies for large roll angles, and the maximum roll angle was calculated $\phi_{max} = 51.0°$, and as the vehicle is symmetric about the x^b-axis, $\phi_{min} = -\phi_{max}$. By using this, we can further limit the range of $c_\theta \in [0.43, \ 1]$.

The simplified 6-DOF MPC control law will thus have the optimization function (4.66), subject to the path constraints (4.68) and the dynamic constraints

$$x(k) = x(k-1) + h\, c_\theta u(k-1) \cos\psi(k-1) \tag{4.72a}$$

$$y(k) = y(k-1) + h\, c_\theta u(k-1) \sin\psi(k-1) \tag{4.72b}$$

$$\psi(k) = \psi(k-1) + h\,\frac{c_\phi}{c_\theta} r \tag{4.72c}$$

$$r(k) = r(k-1) + h\,(\frac{u(k-1)|u(k-1)|\mu_f R \sin\gamma(k-1)}{I_z} - \frac{D}{I_z} r(k-1)|r(k-1)|) \tag{4.72d}$$

4.6 Multiplexed Nonlinear MPC for 3-DOF

We will now derive a MMPC control law for the vehicle in 3-DOF based on the principle's explained in Ling et al. (2011). The idea behind this is to reduce the run time of the controller, which Ling et al. (2011) states that is given by $O((m \times N)^3)$ where m is the number of states, and N is the prediction horizon. Thus, splitting the control law in two optimization formulations might significantly reduce the computation time.

If we consider the control law from **Section 4.4** as a basis, we have $m = 9$ states. By splitting this control law into two optimization formulations where n states are optimized separately, we will have a run time of $O(((m-n) \times N)^3)$ for the first optimization, and $O((n \times N)^3)$ for the second optimization.

One choice we can make here is to optimize the kinematics separately from the yaw kinetics, such that we have one MPC that finds the optimal heading ψ and velocity u to follow the path, whilst the other finds the optimal steering angle γ to reach this heading. By doing this, we will have two controllers, where the input from the outer loop serves as the reference for the inner loop.

We thus have the input $\boldsymbol{\tau} = [u, \ \ \psi_d]^T$, and the outer loop MPC that aims to minimize

$$min \quad \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{P} \\ \boldsymbol{\mathcal{T}} \end{bmatrix}^T \underbrace{\begin{bmatrix} \boldsymbol{Q}_s & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_\epsilon & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \boldsymbol{R} \end{bmatrix}}_{\boldsymbol{G}} \begin{bmatrix} S \\ \boldsymbol{\mathcal{E}} \\ \boldsymbol{P} \\ \boldsymbol{\mathcal{T}} \end{bmatrix} \tag{4.73}$$

subject to

$$x(k) = x(k-1) + hu(k-1)\cos\psi_d \tag{4.74a}$$

$$y(k) = y(k-1) + hu(k-1)\sin\psi_d \tag{4.74b}$$

$$x(k) + \epsilon_x(k) - \lambda_x(k) = 0 \tag{4.74c}$$

$$y(k) + \epsilon_y(k) - \lambda_y(k) = 0 \tag{4.74d}$$

By introducing the states $\boldsymbol{\psi} = [\psi - \psi_d, \ \ r]^T$ and the input γ. We thus have an an inner loop MPC that aims to minimize

$$min \quad \begin{bmatrix} \boldsymbol{\Psi}^T & \Gamma^T \end{bmatrix} \underbrace{\begin{bmatrix} \boldsymbol{Q}_\psi & \mathbf{0} \\ \mathbf{0} & \boldsymbol{Q}_\gamma \end{bmatrix}}_{\boldsymbol{G}_\psi} \begin{bmatrix} \boldsymbol{\Psi} \\ \Gamma \end{bmatrix} \tag{4.75}$$

where

$$\boldsymbol{\Psi} = \begin{bmatrix} \boldsymbol{\psi}(1) \\ \boldsymbol{\psi}(2) \\ \vdots \\ \boldsymbol{\psi}(N) \end{bmatrix} \tag{4.76a}$$

$$\Gamma = \begin{bmatrix} \gamma(1) \\ \gamma(2) \\ \vdots \\ \gamma(N) \end{bmatrix} \tag{4.76b}$$

The weighting matrix $\boldsymbol{Q}_\psi$ will be a $2N \times 2N$ diagonal matrix of Q_ψ where

$$Q_\psi = \begin{bmatrix} q_\psi & 0 \\ 0 & q_r \end{bmatrix} \tag{4.77}$$

As the weight q_ψ aims to converge $\psi - \psi_d \to 0$, whilst q_r converges $r \to 0$, the weights will be chosen such that $q_\psi \gg q_r$. This is done because we only wish to achieve $r = 0$ once the heading has reached the desired heading, which can be enforced by imposing a larger weight on $\psi - \psi_d$ in the optimization. Further, the inner loop optimization will be subject to the constraints

$$\psi(k) = \psi(k-1) + h\, r \tag{4.78a}$$

$$r(k) = r(k-1) + h\,(\frac{u|u|\mu_f R \sin\gamma(k-1)}{I_z} - \frac{D}{I_z} r(k-1)|r(k-1) \tag{4.78b}$$

We thus have two optimization functions with seven and three states respectively, instead of one optimization function with nine states. The total run time of this MMPC should thus be given

$$t = (7 \times N)^3 T + (3 \times N)^3 T = (343N^3 + 27N^3)T = 380N^3 T \tag{4.79}$$

where T is a time unit, i.e the time a computer uses to perform one operation. For comparison, the MPC formulation from **Section 4.4** will have the run time

$$t = (9 \times N)^3 = 729N^3 T \tag{4.80}$$

Chapter 5

Implementation

5.1 Algorithms

5.1.1 Linearizing MPC

We will now derive an algorithm that will realize the linearized MPC. The first step will be the initialization sequence. In the initialization sequence, physical constraints such as $\boldsymbol{\tau_{max}}$, and the weighting matrix $\boldsymbol{G}$ for the objective functions will be defined. These matrices are tuning parameters for the controller, and numerical values for these will be set later. Next measurements of the system states and inputs will be made, and the system will be linearized for these states, as the system dynamics will serve as constraints for the optimization. Lastly, the optimization is solved using a QP-algorithm. The algorithm will thus be as follows

Define: $\boldsymbol{\tau_{max}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{Q}_s$, $\boldsymbol{Q}_\epsilon$ and $\boldsymbol{R}$
while($\boldsymbol{p}^e \neq \boldsymbol{p}_f^e$):
    measure $\boldsymbol{x}(0)$ and $\boldsymbol{\tau}(-1)$
    if($u < 0.1$):
        $u = 0.1$
    calculate A_d and B_d
    solve $min \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z}$ subject to:
        $\boldsymbol{A}_{eq} \boldsymbol{Z} = \boldsymbol{B}_{eq}$
        $\boldsymbol{Z}_{min} \leq \boldsymbol{Z} \leq \boldsymbol{Z}_{max}$
    apply $\boldsymbol{\tau}(0)$

5.1.2 Nonlinear MPC for 3-DOF

The next algorithm we will derive will realize the 3-DOF NMPC. As for the linearizing MPC, the first step of this algorithm will be the initialization sequence, which is identical to that of the linearized MPC. Next, measurements of the system states and inputs will

be made, but rather than linearizing the system with respect to these measurements, the states will be used directly in the nonlinear dynamic equations. Further, these dynamic equations will serve as the MPC's equality constraints together with the path function. As we for the NMPC require to solve a nonlinear optimization problem, a NLP solver needs to be implemented. The solvers used in this thesis will be explained in **Section 5.2**

Define: $\boldsymbol{\tau_{max}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{Q}_s$, $\boldsymbol{Q}_\epsilon$ and $\boldsymbol{R}$
while($\boldsymbol{p}^e \neq \boldsymbol{p}_f^e$):
    measure $\boldsymbol{x}(0)$
    solve $min \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z}$ subject to:
        for($k = 1 \rightarrow N$):
            $\boldsymbol{x}(k) = \boldsymbol{f}(\boldsymbol{x}(k-1), \ \boldsymbol{\tau}(k-1))$
            $\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) - \Lambda(s(k)) = 0$
        $\boldsymbol{Z}_{min} \leq \boldsymbol{Z} \leq \boldsymbol{Z}_{max}$
    apply $\boldsymbol{\tau}(0)$

5.1.3 Nonlinear MPC for 6-DOF

Next we will derive an algorithm that will realize the 6-DOF NMPC. This is almost identical to the algortighm for 3-DOF NMPC, with the only difference being that the pitch and roll angles are measured and included in the dynamic equations. We thus have the algorithm

Define: $\boldsymbol{\tau_{max}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{Q}_s$, $\boldsymbol{Q}_\epsilon$ and $\boldsymbol{R}$
while($\boldsymbol{p}^e \neq \boldsymbol{p}_f^e$):
    measure $\boldsymbol{x}(0)$, $\theta(0)$ and $\phi(0)$
    solve $min \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z}$ subject to:
        for($k = 1 \rightarrow N$):
            $\boldsymbol{x}(k) = \boldsymbol{f}(\boldsymbol{x}(k-1), \ \boldsymbol{\tau}(k-1), \ \theta(0), \ \phi(0))$
            $\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) - \Lambda(s(k)) = 0$
        $\boldsymbol{Z}_{min} \leq \boldsymbol{Z} \leq \boldsymbol{Z}_{max}$
    apply $\boldsymbol{\tau}(0)$

5.1.4 Nonlinear MMPC for 3-DOF

The last algortihms we will derive will realize the 3-DOF MMPC. This control law can be realized in several ways, and we will give two algorithms that might realize this controller. For both algorithms, the initialization sequence and the measurements will be identical to the NMPC for 3-DOF, however two optimization functions will be solved in order. The dynamic equations for position and heading will serve as equality constraints in their respective optimization functions. The first algorithm will solve the outer loop and inner loop for subsequently for each time step, and we thus have the algorithm

Algorithm 1
Define: $\boldsymbol{\tau_{max}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{Q}_s$, $\boldsymbol{Q}_\epsilon$, $\boldsymbol{R}$, $\boldsymbol{Q}_\psi and \boldsymbol{Q}_\gamma$
while($\boldsymbol{p}^e \neq \boldsymbol{p}_f^e$):

measure $X(0)$
solve $min \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z}$ subject to:
for($k = 1 \rightarrow N$):
$\boldsymbol{p}^e(k) = \boldsymbol{f}(\boldsymbol{p}^e(k-1),\ \boldsymbol{\tau}(k-1))$
$\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) - \Lambda(s(k)) = 0$
$\boldsymbol{Z}_{min} \leq \boldsymbol{Z} \leq \boldsymbol{Z}_{max}$
$u(0) = \tau_1(0)$ and $\psi_d = \tau_2(0)$
solve $min\,[\boldsymbol{\Psi}^T,\ \Gamma^T]\boldsymbol{G}_\psi[\boldsymbol{\Psi},\ \Gamma]^T$ subject to:
for($k = 1 \rightarrow N$):
$\boldsymbol{\psi}(k) = \boldsymbol{f}(\boldsymbol{\psi}(k-1),\ \boldsymbol{\gamma}(k-1),\ u(0))$
$\Gamma_{min} \leq \Gamma \leq \Gamma_{max}$
apply $u(0)$ and $\gamma(0)$

The second algorithm will solve the inner and outer loop at different sampling steps. This is done by choosing a larger sampling time for the outer loop than the inner loop, e.g $h_o = 1$s whilst $h_i = 0.1$s as the outer and inner loop sampling time respectively. This will give a piece wise linear desired path for the kinematics, whilst the inner loop aims to reach the desired heading within one outer loop time step. We thus have the algorithm

Algorithm 2
Define: $\boldsymbol{\tau_{max}}$, $\boldsymbol{\Lambda}$, $\boldsymbol{Q}_s$, $\boldsymbol{Q}_\epsilon$, $\boldsymbol{R}$, $\boldsymbol{Q}_\psi and \boldsymbol{Q}_\gamma$
while($\boldsymbol{p}^e \neq \boldsymbol{p}^e_f$):
measure $\boldsymbol{x}(0)$
solve $min \boldsymbol{Z}^T \boldsymbol{G} \boldsymbol{Z}$ subject to:
for($k = 1 \rightarrow N$):
$\boldsymbol{p}^e(k) = \boldsymbol{f}(\boldsymbol{p}^e(k-1),\ \boldsymbol{\tau}(k-1))$
$\boldsymbol{p}^e(k) + \boldsymbol{\epsilon}(k) - \Lambda(s(k)) = 0$
$\boldsymbol{Z}_{min} \leq \boldsymbol{Z} \leq \boldsymbol{Z}_{max}$
$u(0) = \tau_1(0)$ and $\psi_d = \tau_2(0)$
$t = 0$
while($t < h_0$):
solve $min\,[\boldsymbol{\Psi}^T,\ \Gamma^T]\boldsymbol{G}_\psi[\boldsymbol{\Psi},\ \Gamma]^T$ subject to:
for($k = 1 \rightarrow N$):
$\boldsymbol{\psi}(k) = \boldsymbol{f}(\boldsymbol{\psi}(k-1),\ \boldsymbol{\gamma}(k-1),\ u(0))$
$\Gamma_{min} \leq \Gamma \leq \Gamma_{max}$
apply $u(0)$ and $\gamma(0)$
$t = t + h_i$

5.2 Solvers and hardware

For this thesis, we will use two solver modules for the NLP optimization. The first module is APMonitor which, is a free dynamic optimization toolbox for MATLAB. The APMonitor toolbox was donwloaded from http://apmonitor.com/wiki/index.php/Main/MATLAB, and optimization problems are solved online on APMonitor's servers. APMonitor offers three solvers, and for this thesis we will use the Active Set solver APOPT, and the Interior

Point (IP) solver IPOPT.

The second solver module we will use is SciPy, which is a free toolbox for Python. The SciPy toolbox contains solvers for dynamic optimization, and for this thesis we will use the Active Set solver SQP.

The simulations will be tested on two different computers. Mainly, the simulations will be done on a computer with Intel Core i7-6700HQ processor. This processor has eight cores, and a clock speed of 2.60GHz. Further this machine has 16GB available Random Access Memory (RAM).

The second computer will be a Virtual Machine (VM) run on the aforementioned computer. The VM will therefore have the same processor, however it will be limited to four out of eight cores. Further, the computer will only have 4GB available RAM

Chapter 6

Results

6.1 Basic optimization

Before testing the different MPC control laws, we will first test the basic optimization formulation derived in **Section 4.2**, as all the control laws are based on this optimization formulation. In order to test the functionality of this optimization, a simple simulation is made in MATLAB. For this simulation, we will only use the simple dynamics

$$\dot{x}^e = \tau_1 \tag{6.1a}$$

$$\dot{y}^e = \tau_2 \tag{6.1b}$$

It is important to note here that these dynamics does not represent the actual dynamics of the UGV, but is just introduced to test the theory of the new optimization formulation. The results should therefore not be taken as a guarantee for the performance of the actual controller, however the results will give an indication on whether this optimization formulation could work as a basis for the control laws, i.e converging to the path before converging towards the end point. The desired path will be the same as for (4.4), whilst the initial position will still be $\boldsymbol{p}^e(0) = (9, \ \ 2.5)^T$. The prediction horizon is chosen $N = 10$ and the system is simulated for 50 time steps, with the sample time $h = 0.1\text{s}$. For this simulations we chose the weighting and cost matrices with $q_\epsilon = 1000$, $q_e = 1$ and $r = 0.1$, with both inputs bounded, i.e $-4 \leq \boldsymbol{\tau} \leq 4$. **Fig. 6.1** shows the results of the simulation.

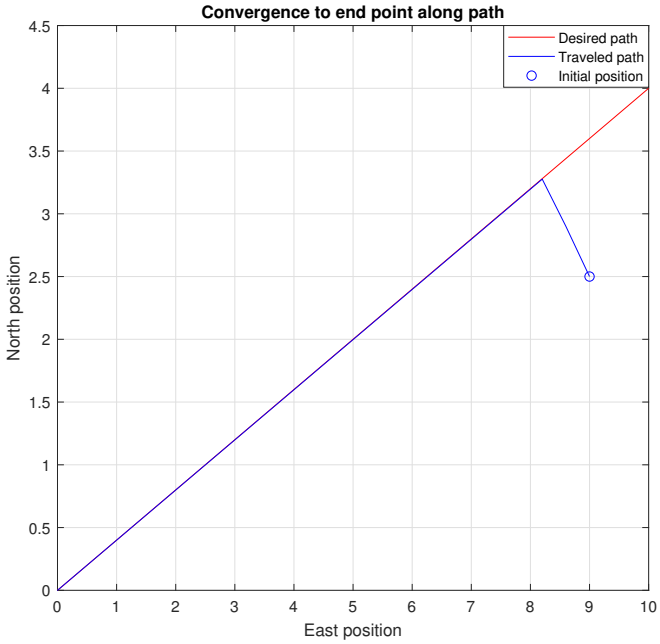

Figure 6.1: Simulation of a simple path following example

As we can see, this control law is able to make the object converge to the path, before following the path towards the end point. We note here that the control law drives the object towards the path point $\boldsymbol{p}_d^e = [8.20 \;\; 3.28]^T$ instead of the optimal path point $\boldsymbol{p}_d^e = [8.62 \;\; 3.45]^T$. However when comparing the initial errors for these path points, we have an error of $e = 1.25$, whilst the error from the optimal path position is $e = 1.05$. The algorithm thus converges towards $\boldsymbol{p}_d^e = [8.20 \;\; 3.28]^T$, as this will also minimize the error from the end point. I.e, the optimization control law converges towards a point that is slightly further away from the initial position, in order to minimize the overall objective function. As the control law achieved the desired goal, we will move further with the control laws for the UGV.

6.2 Linearized MPC for 3-DOF

For simplicity, the linearized MPC was first tested with the linear path, i.e $\boldsymbol{\Lambda}_1$. The simulations where done in MATLAB using MATLAB's QP solver "quadprog", and the first tests were done with a prediction horizon $N = 30$ and a simulation time of 30 seconds. The controller was simulated with the linear damping model and no disturbances, simulating a perfect model. The weighting matrices where chosen as

$$q_\epsilon = 1000, \qquad R = \begin{bmatrix} 0.1 & 0 \\ 0 & 1 \end{bmatrix}$$

and with $q_s = 1$, $q_s = 10$, $q_s = 100$ and $q_s = 1000$, with results shown in respective order in **Fig. 6.2** to **Fig. 6.5**

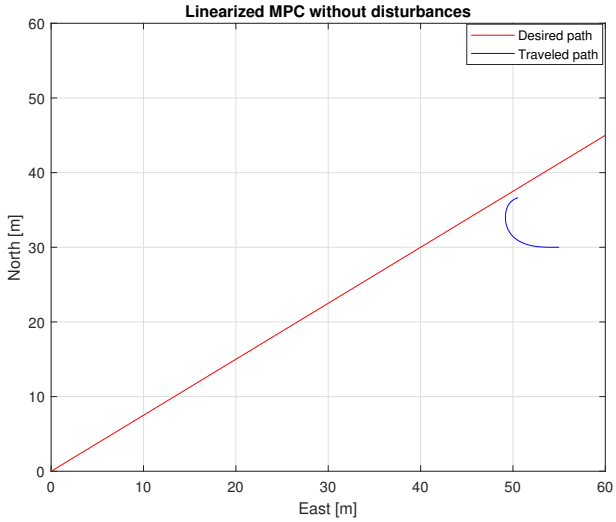

Figure 6.2: Simulation of linearized MPC without disturbances and $q_s = 1$

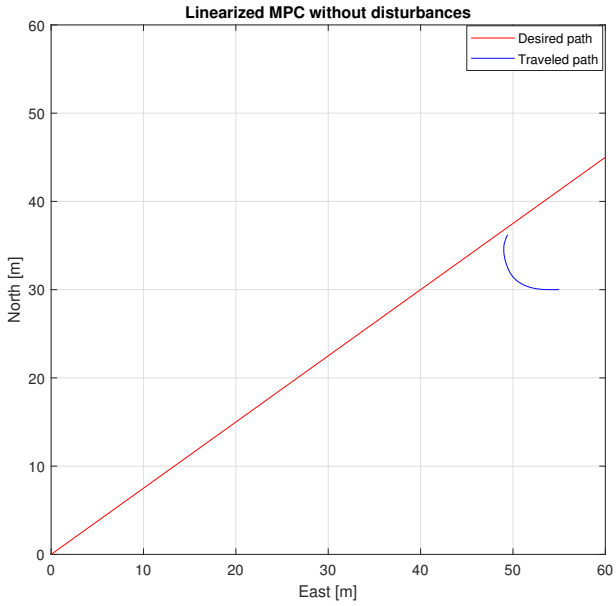

Figure 6.3: Simulation of linearized MPC without disturbances and $q_s = 10$

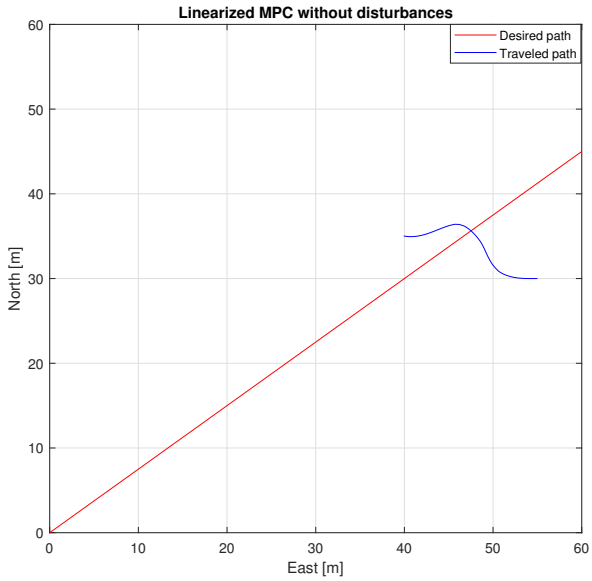

Figure 6.4: Simulation of linearized MPC without disturbances and $q_s = 100$

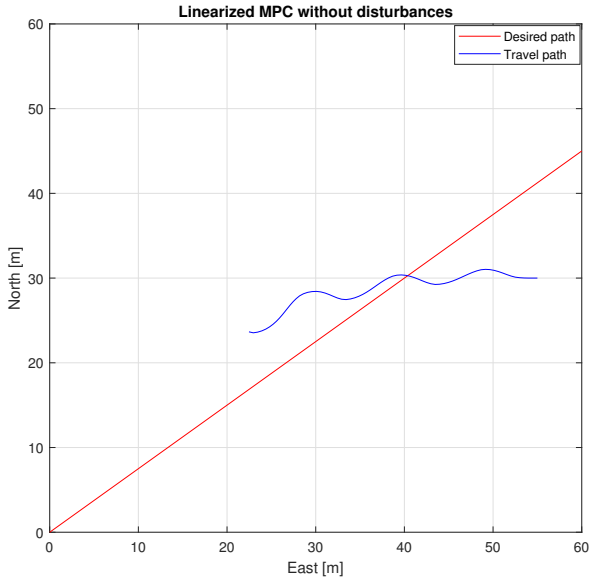

Figure 6.5: Simulation of linearized MPC without disturbances and $q_s = 1000$

As we can see, none of these simulations were able to converge to the final waypoint with the linearized MPC. The final velocity was zero for all controllers, meaning the vehicle had come to a rest at the end of the simulation. **Fig. 6.6** shows that the velocity input from the MPC is zero from eleven seconds of the simulation, which implies that the MPC control law is unable to converge the vehicle any further towards the end.

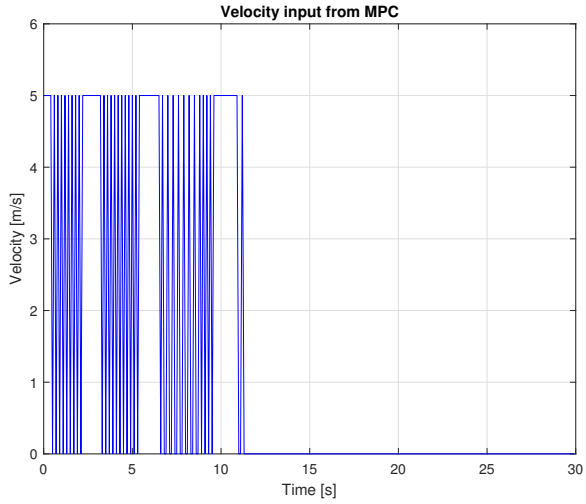

Figure 6.6: MPC velocity input with $q_s = 1000$

In an attempt to make the simulated UGV converge to the end point, two modifications were made to the MPC. First, the weighting matrix was changed such that

$$R = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix} \tag{6.2}$$

which removes the penalty on the use of input in the control law. The motivation behind this is that if $\boldsymbol{\mathcal{T}}^T \boldsymbol{R} \boldsymbol{\mathcal{T}} > S^T \boldsymbol{Q}_s S$, this would imply that the cost of using a input exceeds that penalty imposed on deviations from the final waypoint. Thus, by removing the cost of using an input, the MPC can use an input freely as long as it does not violate any constraints or as long as the simulated UGV has not reached the final waypoint.

The second modification that was made was an increase of prediction horizon such that $N = 100$. By increasing the prediction horizon, the MPC might find some solutions that are impossible for smaller prediction horizons. Consider an UGV facing away from the desired path direction. In order to turn the vehicle, we require that the vehicle has a velocity, however this would move the vehicle further away from the end point, and thus increasing the total cost of the objective function. For prediction horizons shorter than the number of time steps required to turn the vehicle, this would imply that the total function value of the objective function increases, and is thus disregarded as a solution. However for longer prediction horizons, the total function value will decrease as the control law is able to calculate function values after the vehicle has turned. Thus the prediction horizon

was increased in an attempt to converge towards the final waypoint. The results of the simulation with the weight matrix from (6.2) and prediction horizon $N = 100$ is shown in **Fig. 6.7** whilst the velocity input is shown in **Fig. 6.8**

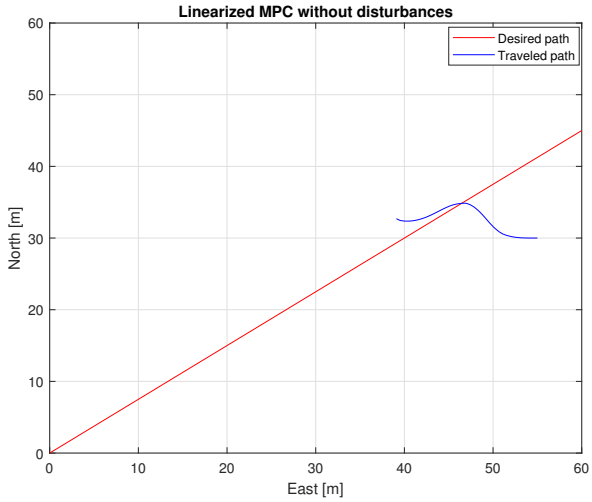

Figure 6.7: Simulation of linearized MPC without disturbances and $q_s = 100$

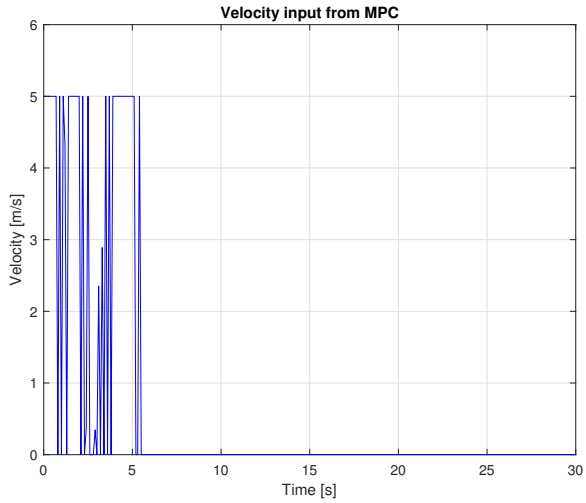

Figure 6.8: MPC velocity input with $q_s = 100$

As we can see, the linearized MPC was still unable to converge the UGV along the path to the end point, and the simulated UGV stops after about five seconds of the simulation.

6.3 Nonlinear MPC for 3-DOF

6.3.1 APMonitor solvers

We will now test the NMPC control law from **Section 4.4**. The first simulation was done for 35 seconds in MATLAB using the APMonitor solver with a prediction horizon of $N = 40$. The desired path was $\mathbf{\Lambda}_2$ and the states are initialized at $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{\pi}{2}, 0]^T$. The system was simulated without disturbances. This will represent a system where the UGV is modelled perfectly, and is unaffected by external forces. The purpose of this simulation is to give an indication on whether the control law works under ideal circumstances, as we will not move further with this controller if this does not perform satisfactory. The results are shown in **Fig. 6.9**.

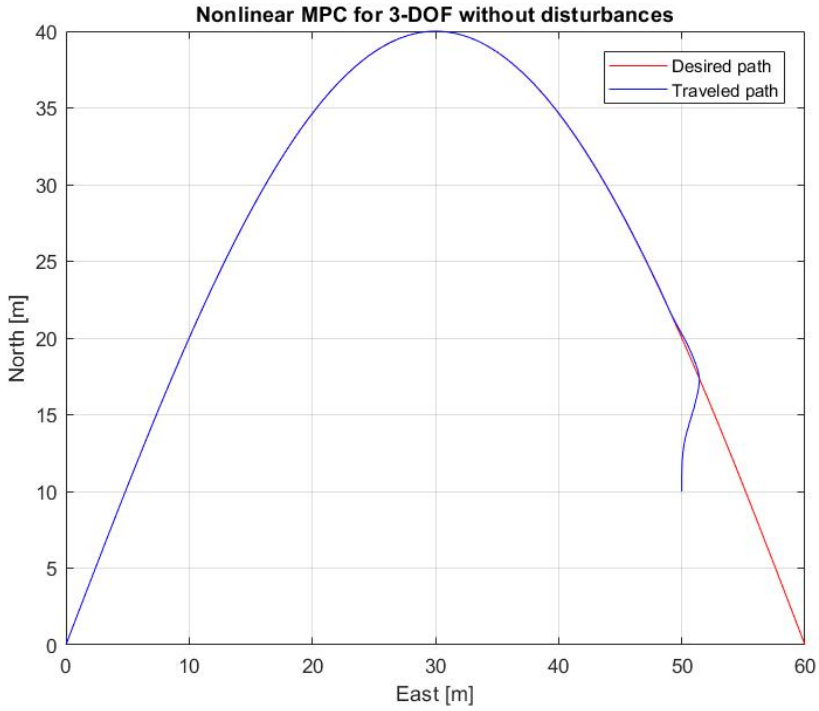

Figure 6.9: Simulation of the NMPC for 3-DOF with ideal model

Further the same simulation was done with the initial states $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{3\pi}{2}, 0]^T$, which implies that the vehicles start at the same position as the previous simulation, but with the heading turned $180°$. The results are shown in **Fig. 6.10**

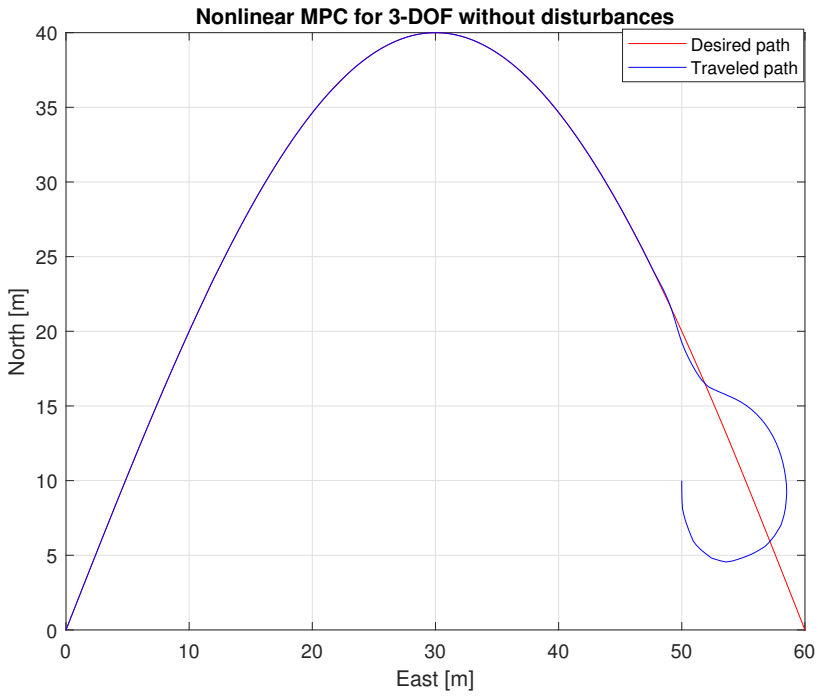

Figure 6.10: Simulation of the NMPC for 3-DOF with ideal model and initial heading away from the path direction

As we can see, the MPC is able to converge the simulated vehicle to the desired path for both simulations, before following the path towards the end point. Further, the velocity at the final iteration of the simulation is $u = 0$, which implies that the vehicle has come to rest at the final waypoint.

Next, the same controller was simulated with disturbances affecting the system. This was done by implementing a time variant random number, i.e varying for each iteration, to simulate the pitch and roll where $\theta \in [0°,\ 20°]$ and $\phi \in [0°,\ 20°]$. This was used to implement the kinematic equations from (3.2). Negative angles are ignored, as the kinematics are given by $\cos\theta$ and $\cos\phi$, as $\cos -x = \cos x$. Further, two random numbers which was varying for every iteration, $\delta_D \in [0.8,\ 1.2]$ and $\delta_\mu \in [0.8,\ 1.2]$ was introduced in the yaw dynamics to simulate model uncertainties in the damping and friction coefficients respectively. These parameters will simulate a 20% uncertainty in the coefficients, which will vary depending on the surface, i.e $D = D_0 \cdot \delta_D = 1528 \cdot \delta_D$ and $\mu_f = \mu_{f0} \cdot \delta_\mu = 37.9 \cdot \delta_m u$. The simulation was again run for the path $\mathbf{\Lambda}_2$ and the states are initialized at $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{\pi}{2}, 0]^T$ and $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{3\pi}{2}, 0]^T$. The results are shown in **Fig. 6.11** and **Fig. 6.12**

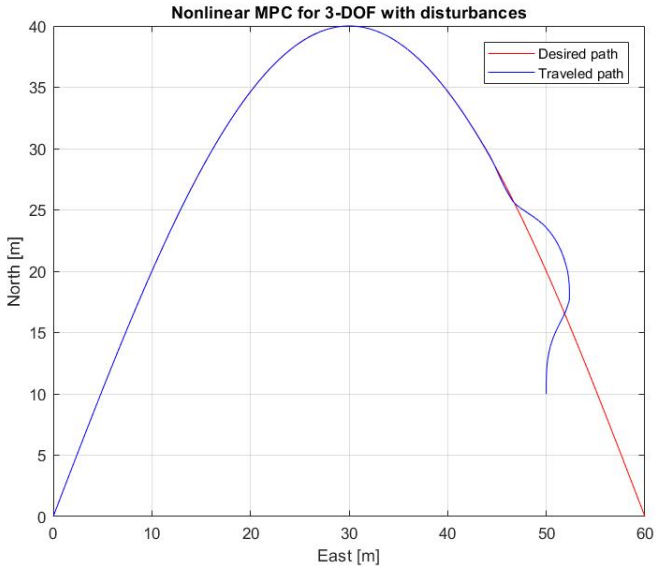

Figure 6.11: Simulation of the NMPC for 3-DOF with model disturbances and initial heading towards the path direction

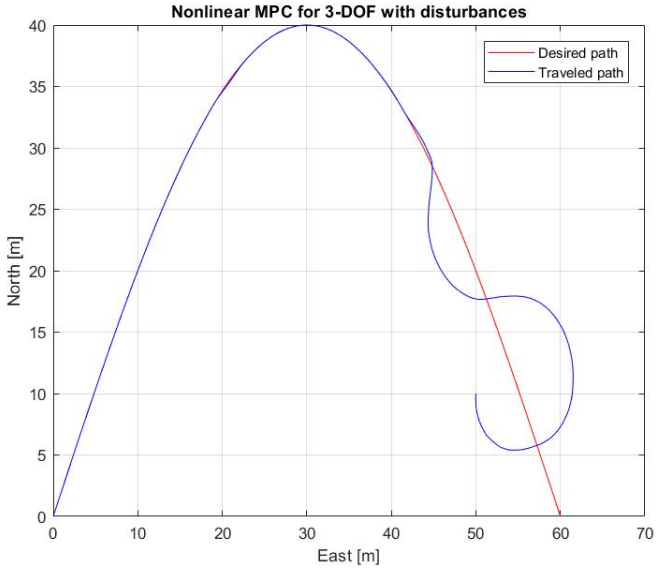

Figure 6.12: Simulation of the NMPC for 3-DOF with model disturbances and initial heading away from the path direction

The simulations were also run with the APMonitors Active Set solver APOPT, and a comparison of the results for the APOPT and IPOPT solver for initial condition $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]$ $[50,\ 10,\ \frac{\pi}{2}, 0]^T$ are shown in **Fig. 6.13**. As we can see, there is no noticeable difference in performance for the two solvers.

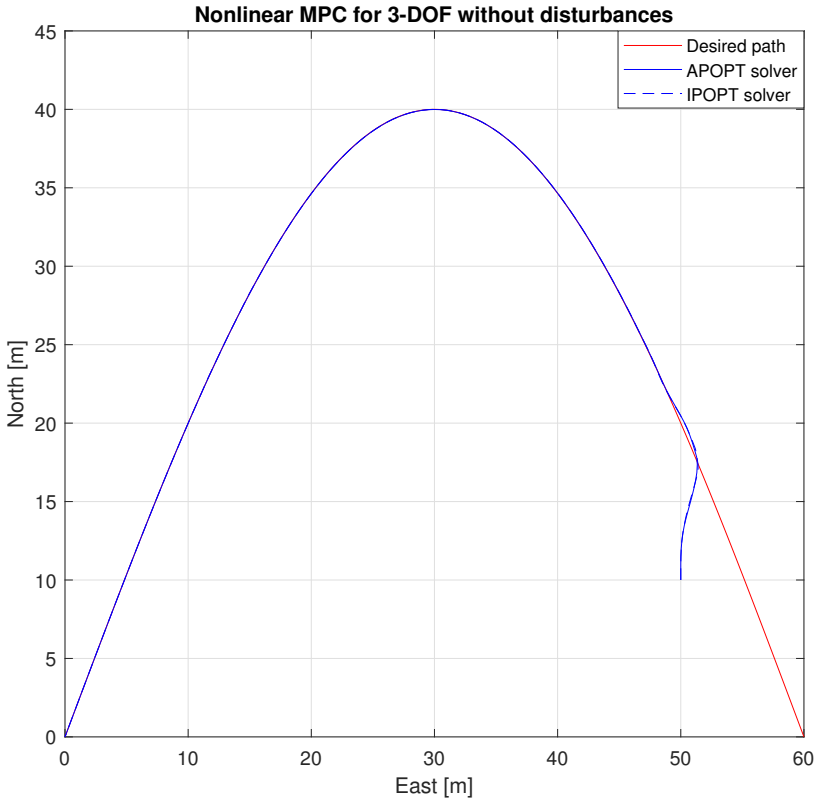

Figure 6.13: Comparison of APOPT and IPOPT solver for 3-DOF Nonlinear MPC

Next, the same MPC was tested for the linear path $\mathbf{\Lambda}_1$. The first simulation for this path was run with a prediction horizon $N = 10$, and initial states $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [60,\ 50,\ \pi,\ 0]^T$. The system was simulated without disturbances, and with a perfect model, and the result are shown in **Fig. 6.14**

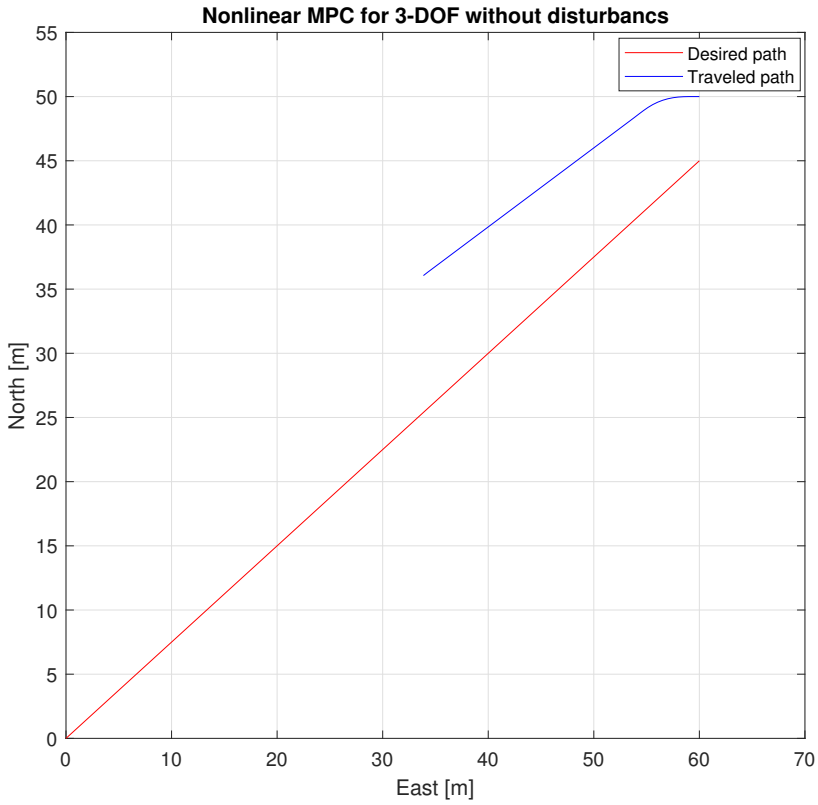

Figure 6.14: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 10$

As we can see, this MPC has neither reached the path nor the final waypoint at the end of the simulation. The prediction was thus increased to $N = 30$, which was the shortest prediction horizon able to follow the path $\mathbf{\Lambda}_2$ for all initial positions. The system was thus simulated for the initial positions $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 30,\ \frac{\pi}{2}, 0]^T$, $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 30,\ \frac{3\pi}{2}, 0]^T$ and $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [65,\ 50,\ \pi, 0]^T$, both with and without disturbances, and the results are shown in respective order in **Fig. 6.15** to **Fig. 6.20** in respective order. The disturbances was implemented as for **Fig. 6.11** and **Fig. 6.12**

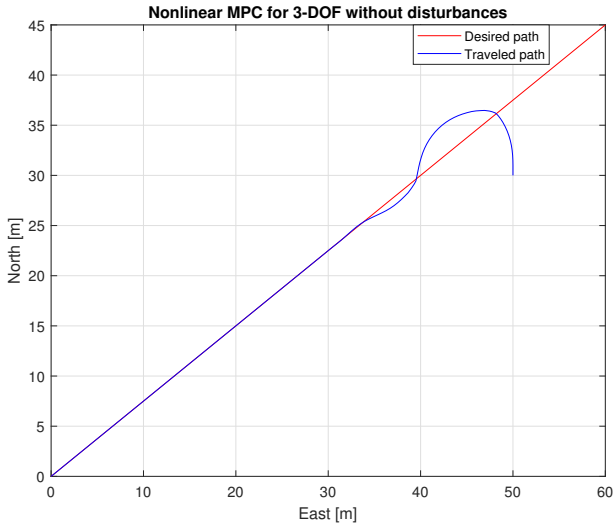

Figure 6.15: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances

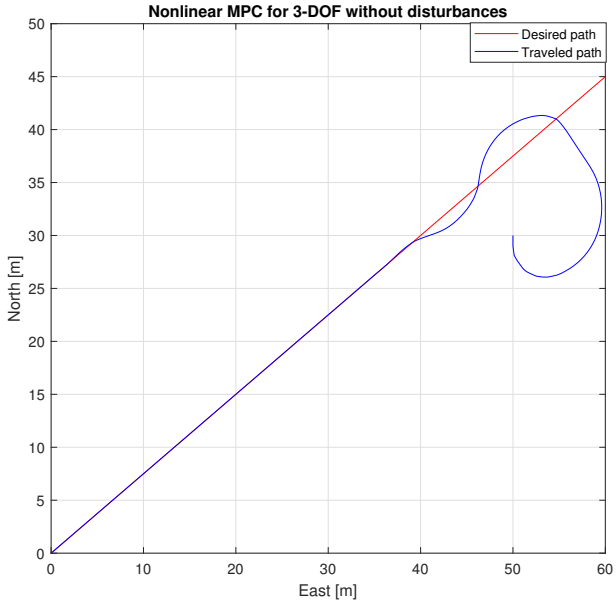

Figure 6.16: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances

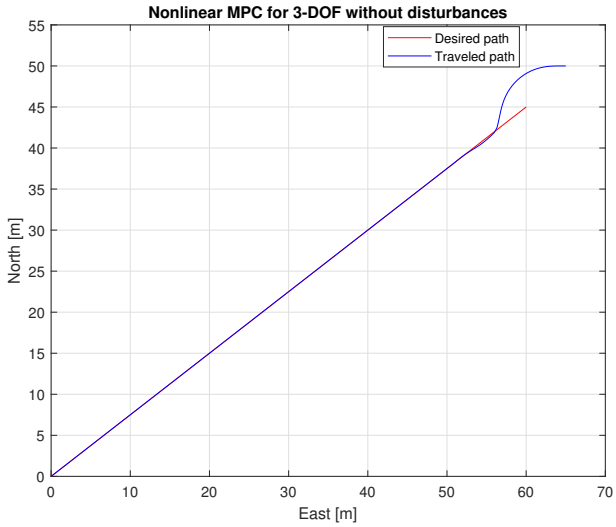

Figure 6.17: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and without disturbances

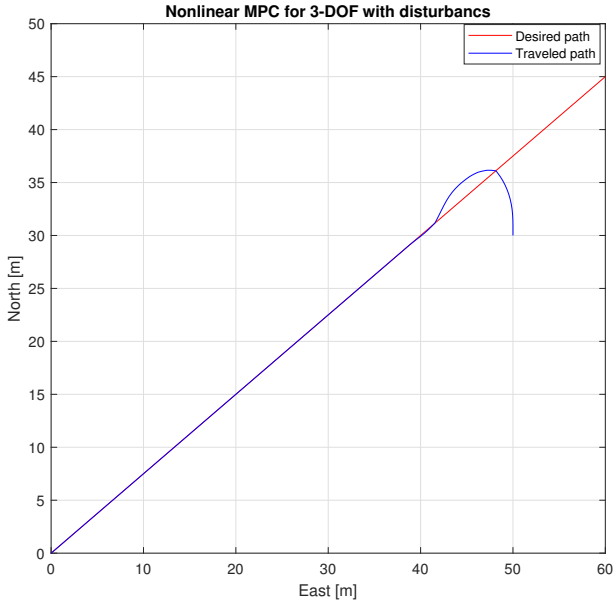

Figure 6.18: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and with disturbances

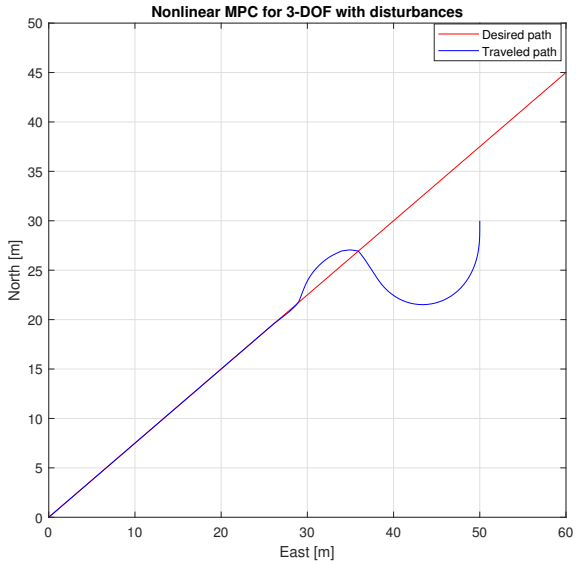

Figure 6.19: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$ and with disturbances

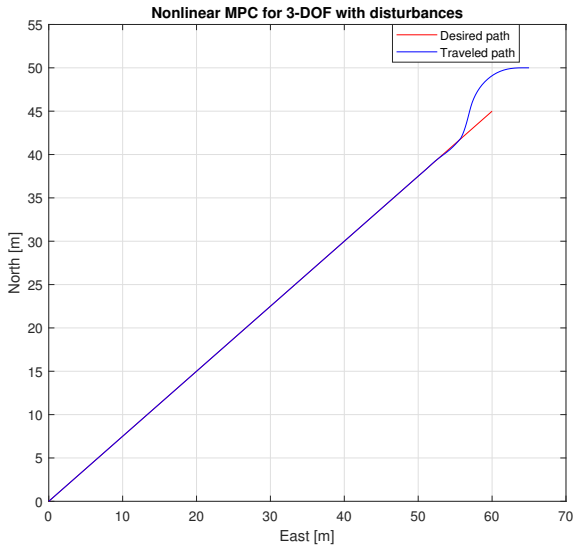

Figure 6.20: Simulation of the NMPC for 3-DOF with linear path and ideal model with $N = 30$

6.3.2 SciPy solver

As the control law was able to follow the path using both APMonitors IP and Active Set solver, an implementation was made in Python using the SciPy SQP solver. The system was simulated with the same desired path and initial state as **Fig. 6.9**. The simulations were run with prediction horizon $N = 10$, $N = 20$, $N = 30$ and $N = 40$, and the results are shown in **Fig. 6.21** to **Fig. 6.24** in respective order.

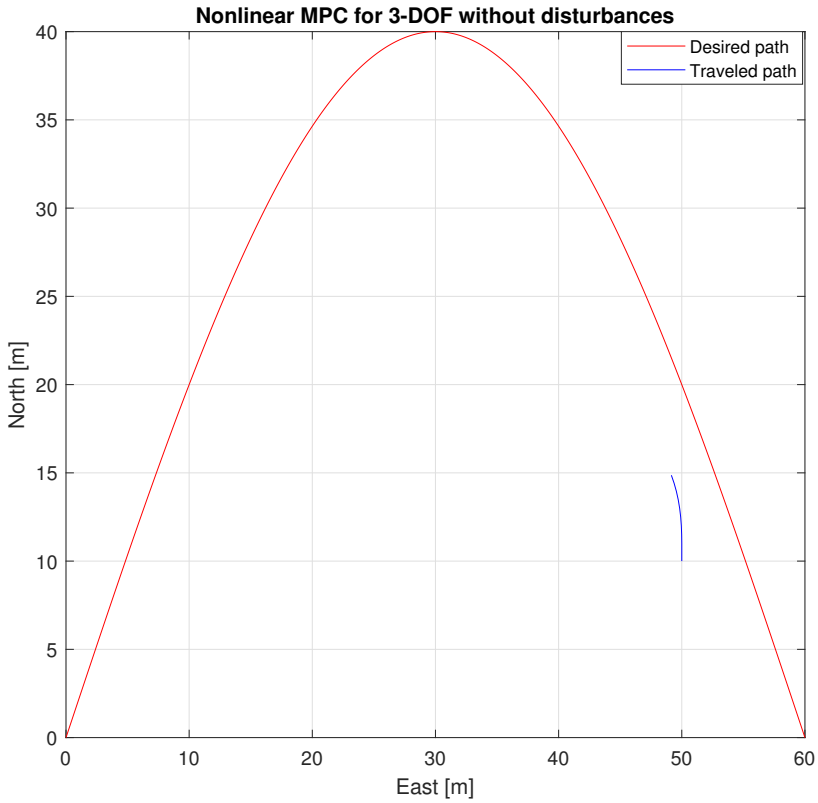

Figure 6.21: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 10$ without disturbances

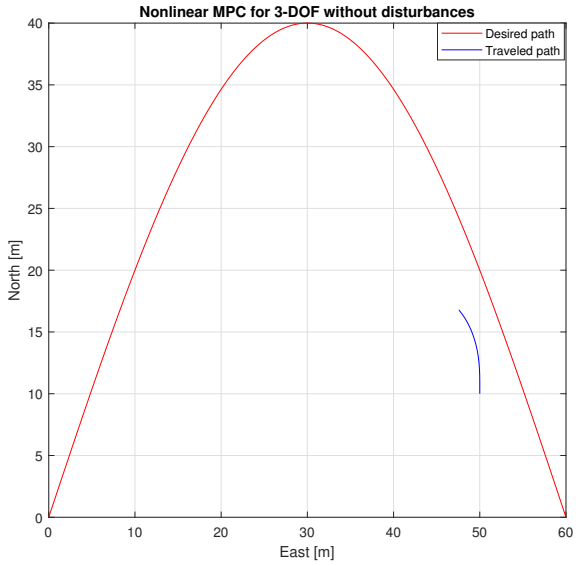

Figure 6.22: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 20$ without disturbances

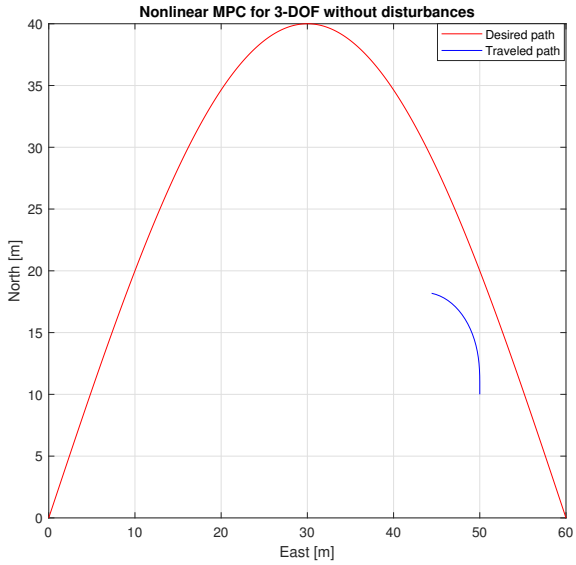

Figure 6.23: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 30$ without disturbances

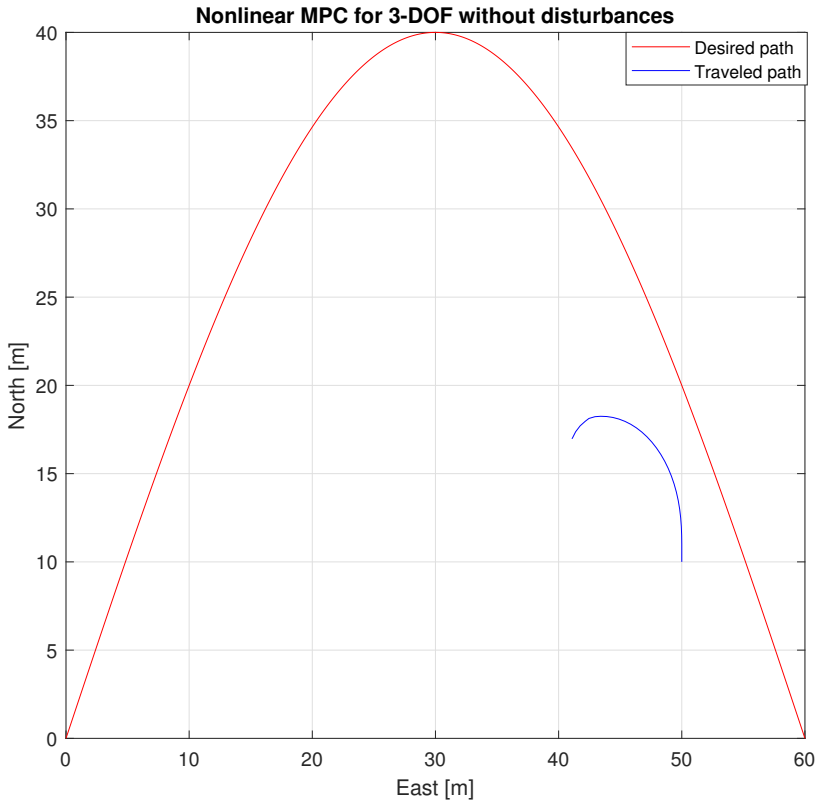

Figure 6.24: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 40$ without disturbances

As we can see, the implementation of the control law with SciPy's SQP solver was unable to both converge to the path and follow the path $\mathbf{\Lambda}_2$ for all prediction horizons. Further, the control law with the SciPy controller was tested for $\mathbf{\Lambda}_1$, with initial condition $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [65,\ 50,\ \pi, 0]^T$. The results for $N = 10$, $N = 20$, $N = 30$, $N = 40$ are shown in **Fig. 6.25** to **Fig. 6.28** in respective order.

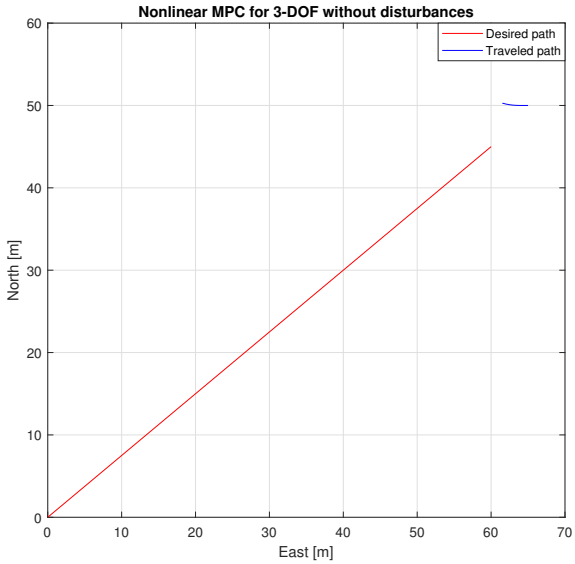

Figure 6.25: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 10$ with linear path and no disturbances

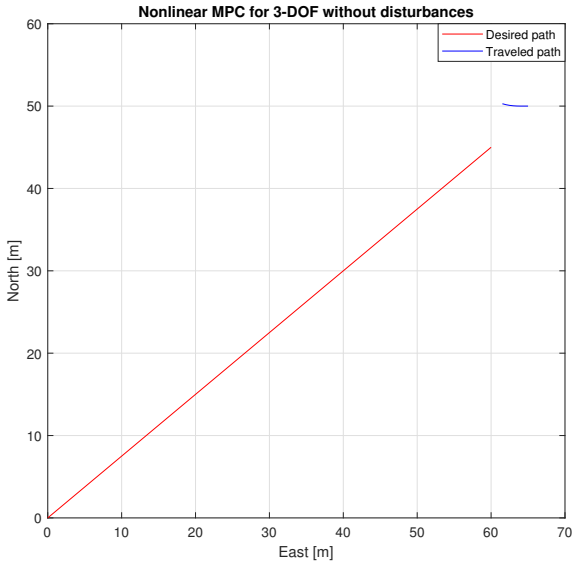

Figure 6.26: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 20$ with linear path and no disturbances

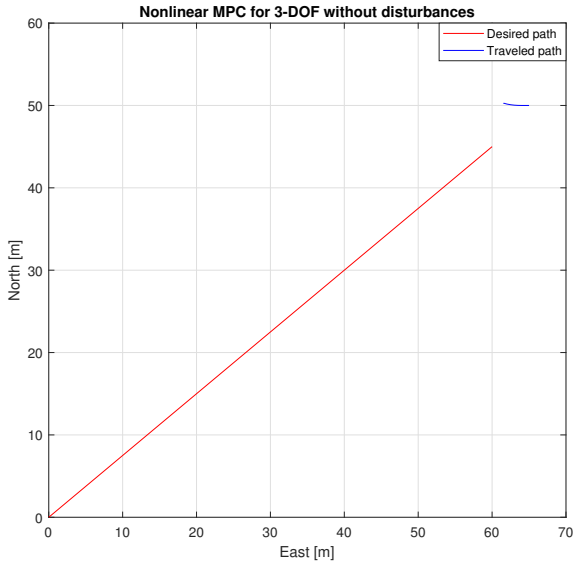

Figure 6.27: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 30$ with linear path and no disturbances

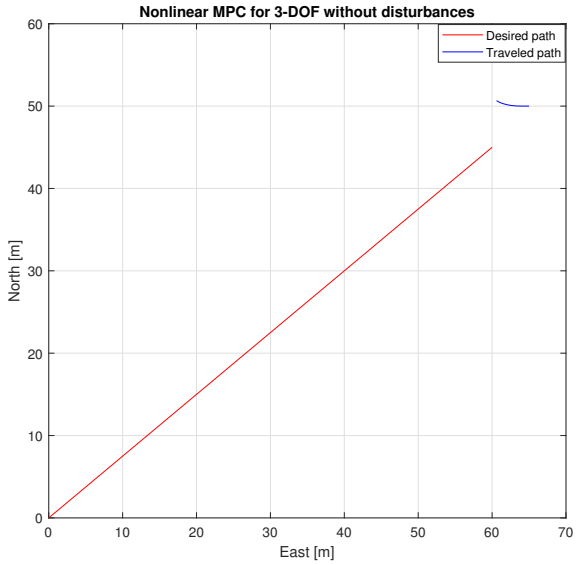

Figure 6.28: Simulation of the NMPC for 3-DOF with SciPy SQP solver with $N = 40$ with linear path and no disturbances

6.4 Simplified 6-DOF controller

The simplified 6-DOF controller was simulated and compared to the 3-DOF controller. This was done with APMonitor's APOPT solver in a MATLAB. The first simulation was made with a constant orientation with $\theta = 30°$, $\phi = 0°$ and zero uncertainty for the damping and friction coefficients. The system was simulated for 35 seconds with a prediction horizon of $N = 40$ time steps, and initial states $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{\pi}{2}, 0]^T$. **Fig. 6.29** shows a comparison of the two control laws.

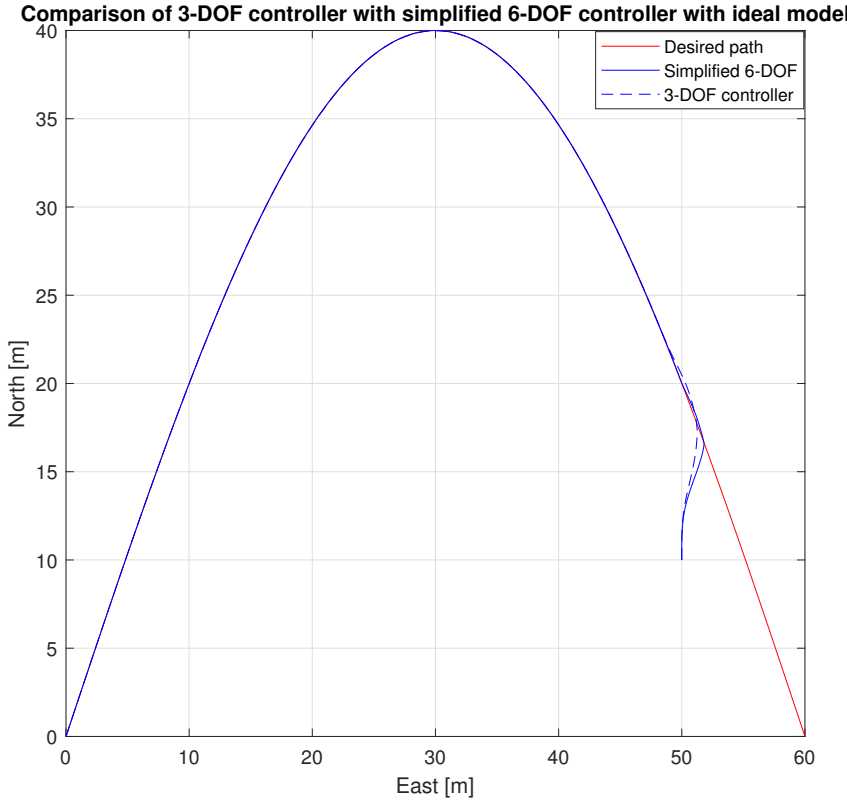

Figure 6.29: Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$ and no disturbances

Further, the two controllers was simulated again using the same model, from the initial point $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [60,\ 10,\ \frac{\pi}{2}, 0]^T$, and the results are shown in **Fig. 6.30**

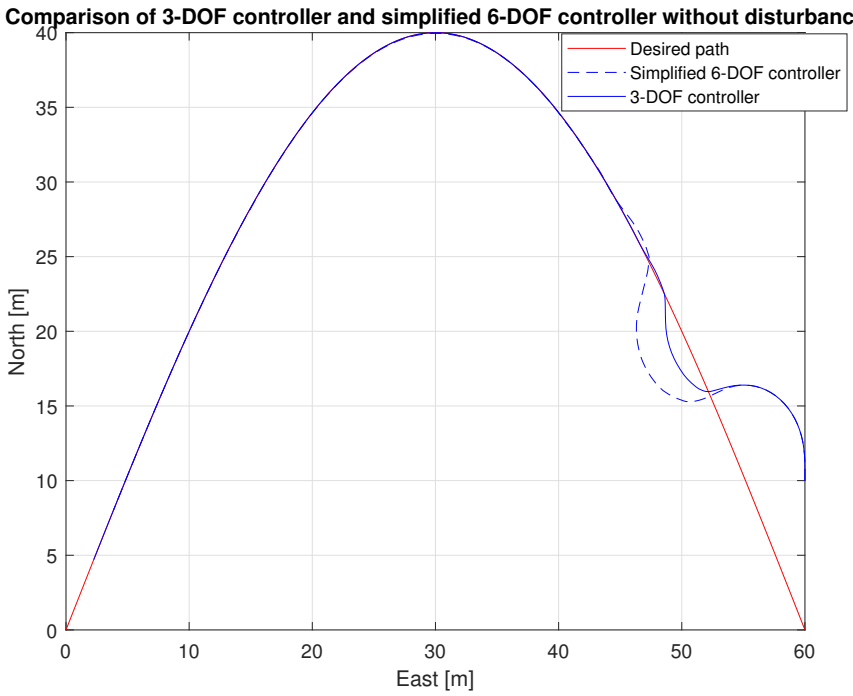

Figure 6.30: Comparison of 3-DOF controller and enhanced 3-DOF controller with $\theta = 30^\circ$, starting in $(60,\ 10)$

In order to test the functionality of the control laws under imperfect conditions, both controllers were simulated again with a side slip. The side slip was introduced in the simulations such that the UGV slides sideways whilst turning, and is caused by the centripetal forces acting on the vehicle. Thus the system is simulated using the model

$$\dot{x}^e = h\,u\cos\psi + h\,c_1\,r\sin\psi \tag{6.3a}$$

$$\dot{y}^e = h\,u\sin\psi + h\,c_1\,r\cos\psi \tag{6.3b}$$

where the parameter c_1 can be changed to reduce or increase the effect of the side slip. For the first simulations, $c_1 = 1$ was chosen. The system was thus simulated with initial states $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{\pi}{2}, 0]^T$, and $\delta_D = \delta_\mu = 1$, and the results are shown in **Fig. 6.31**

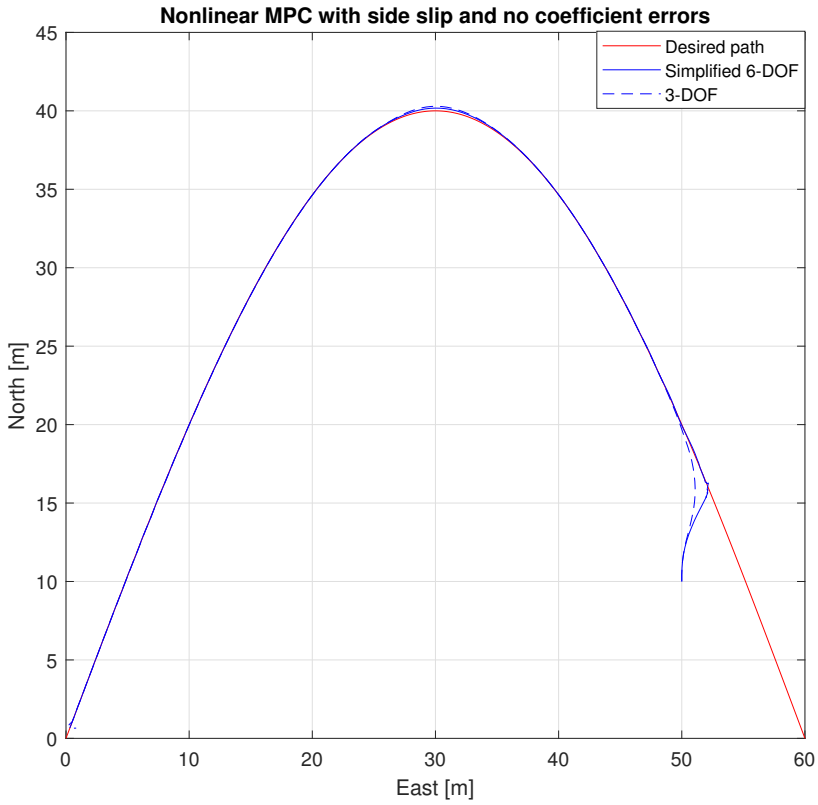

Figure 6.31: Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect on the vehicle

Next, the simulations was run again with errors in the damping and friction coefficients. **Fig. 6.32** and **Fig. 6.33** shows the simulation results with $\delta_D \in [0.8,\ \ 1.2]$ and $\delta_\mu \in [0.8,\ \ 1.2]$, and $\delta_D \in [1,\ \ 1.2]$ and $\delta_\mu \in [1,\ \ 1.2]$ respectively.

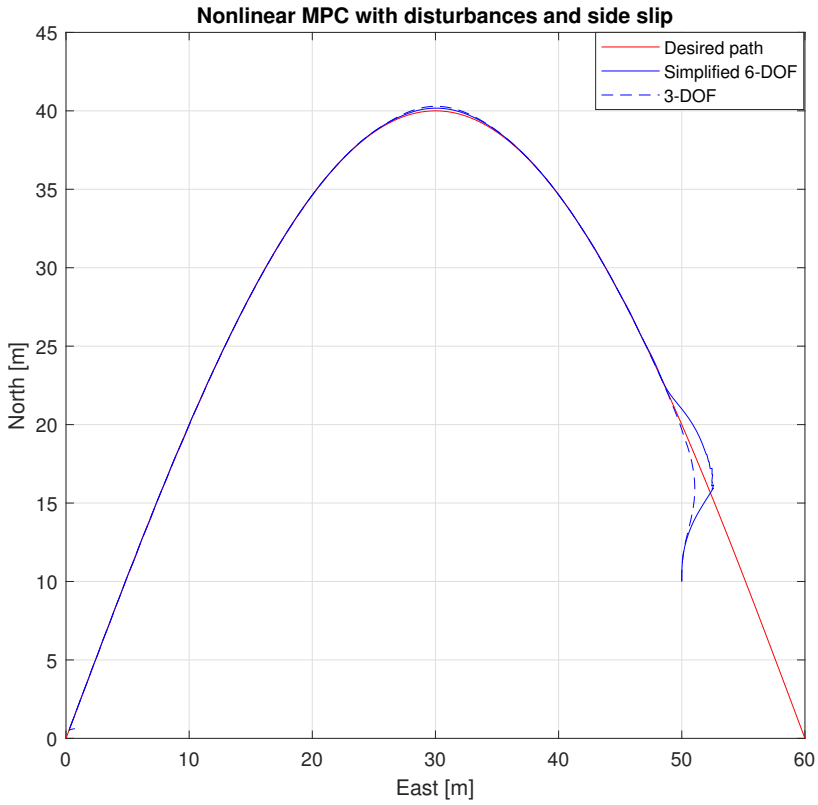

Figure 6.32: Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect and coefficient errors $\delta \in [1, \quad 1.2]$

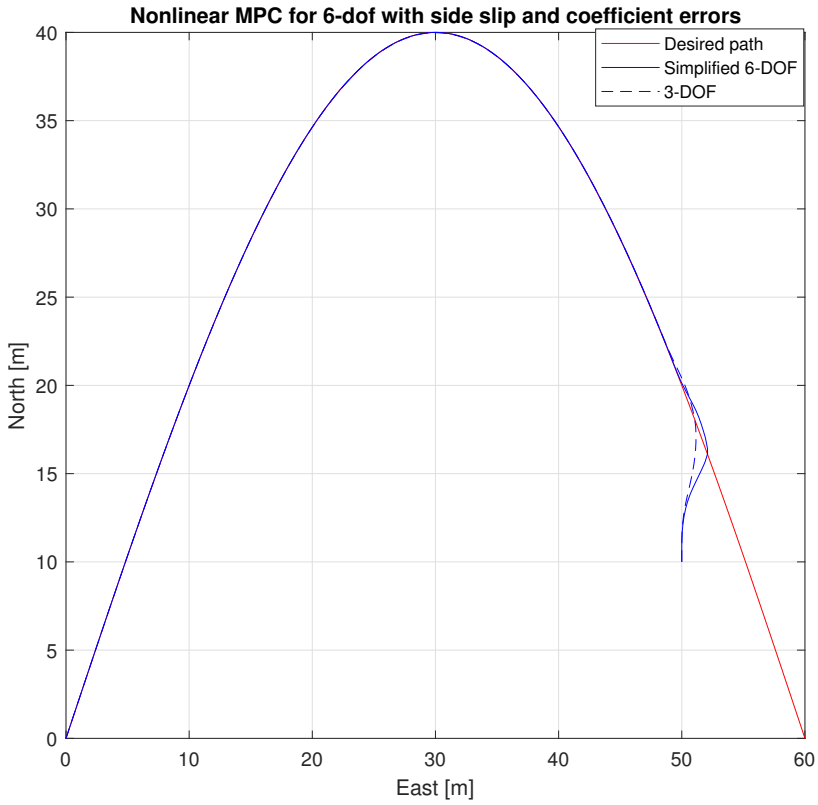

Figure 6.33: Comparison of 3-DOF controller and simplified 6-DOF controller with $\theta = 30°$, with a side slip effect and coefficient errors $\delta \in [1, \quad 1.2]$

The simplified 6-DOF MPC was also simulated with a larger side slip, i.e $c_1 = 10$, with the coefficient errors $\delta_D \in [0.8, \quad 1.2]$ and $\delta_\mu \in [0.8, \quad 1.2]$. The prediction horizon was chosen as $N = 40$, and the results are shown in **Fig. 6.34**.

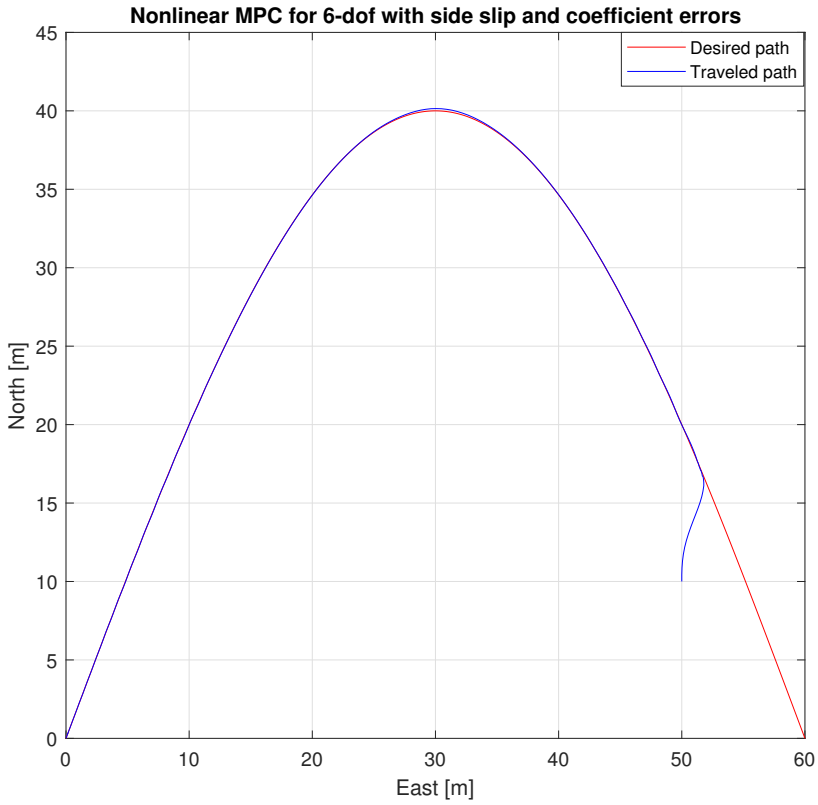

Figure 6.34: Simplified 6-DOF MPC with large side slip

6.5 Multiplexed Model Predictive Control

6.5.1 APMonitor solver

The MMPC control law was first simulated with the APMonitor APOPT solver. The prediction horizon was chosen as $N = 40$ and the path $\mathbf{\Lambda}_2$ was chosen. Both the MMPC algorithms was run for initial position $[x^e(0),\ \ y^e(0),\ \ \psi(0),\ \ r(0)]^T = [50,\ \ 10,\ \ \frac{\pi}{2}, 0]^T$ for 35 seconds, and the results are shown in **Fig. 6.35**

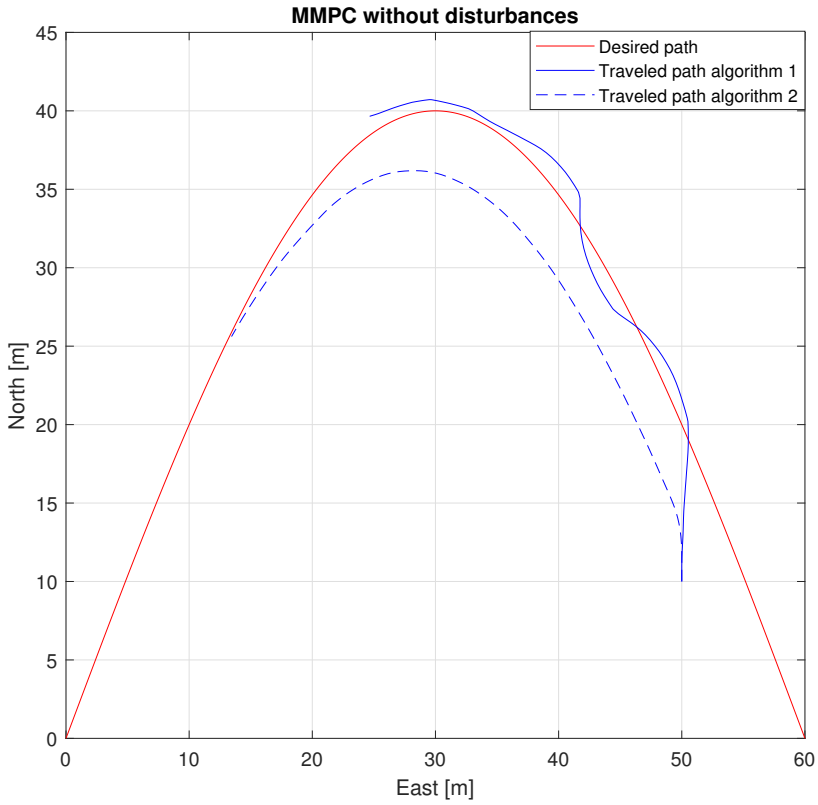

Figure 6.35: Comparison of algorithm 1 and 2 for Multiplexed Model Predictive Control

6.5.2 SciPy solver

Next the MMPC was simulated for $\mathbf{\Lambda}_2$ with the SciPy solver, for initial states chosen as $[x^e(0),\ y^e(0),\ \psi(0),\ r(0)]^T = [50,\ 10,\ \frac{\pi}{2}, 0]^T$ without disturbances. The MMPC was simulated with both algorithm 1 and algortihm 2 with prediction horizon $N = 10$, $N = 20$, $N = 30$ and $N = 40$, and the same results. The results are shown in **6.36** to **6.39** in respective order.

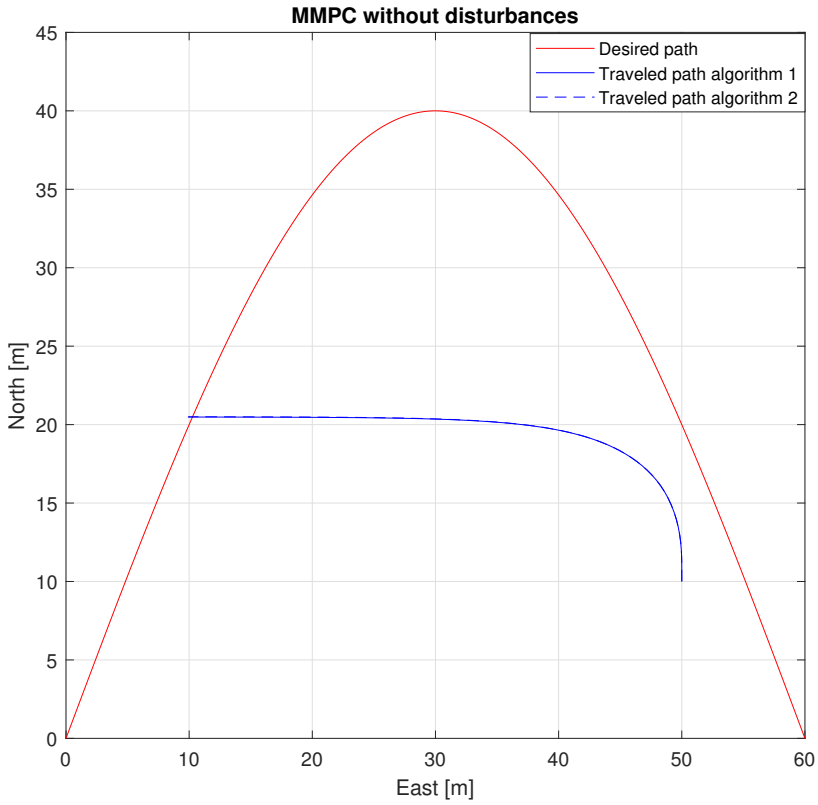

Figure 6.36: Multiplexed Model Predictive Control without disturbances with $N = 10$

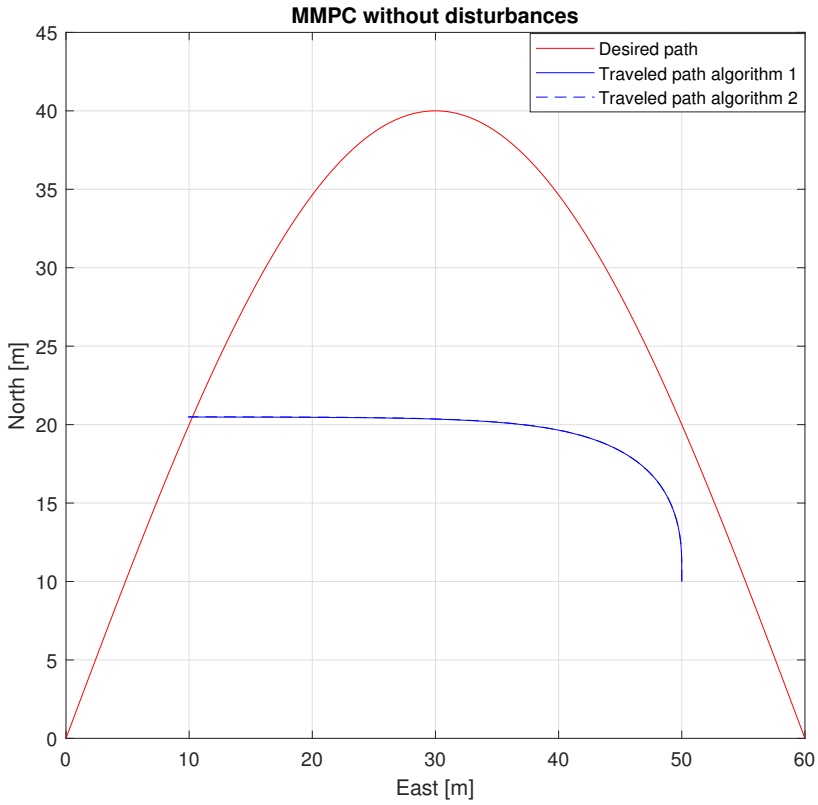

Figure 6.37: Multiplexed Model Predictive Control without disturbances with $N = 20$

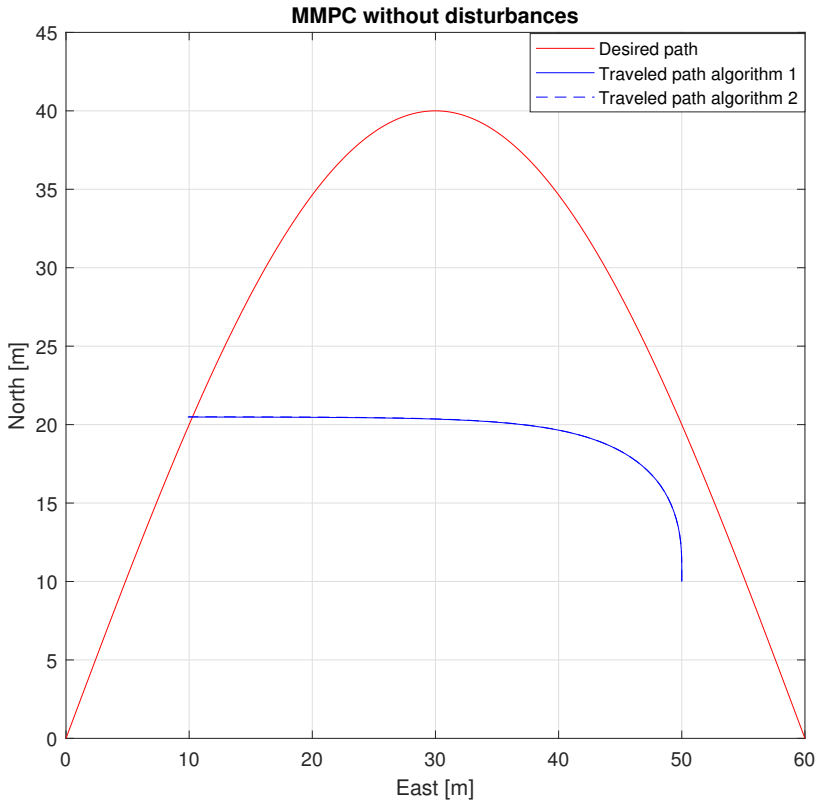

Figure 6.38: Multiplexed Model Predictive Control without disturbances with $N = 30$

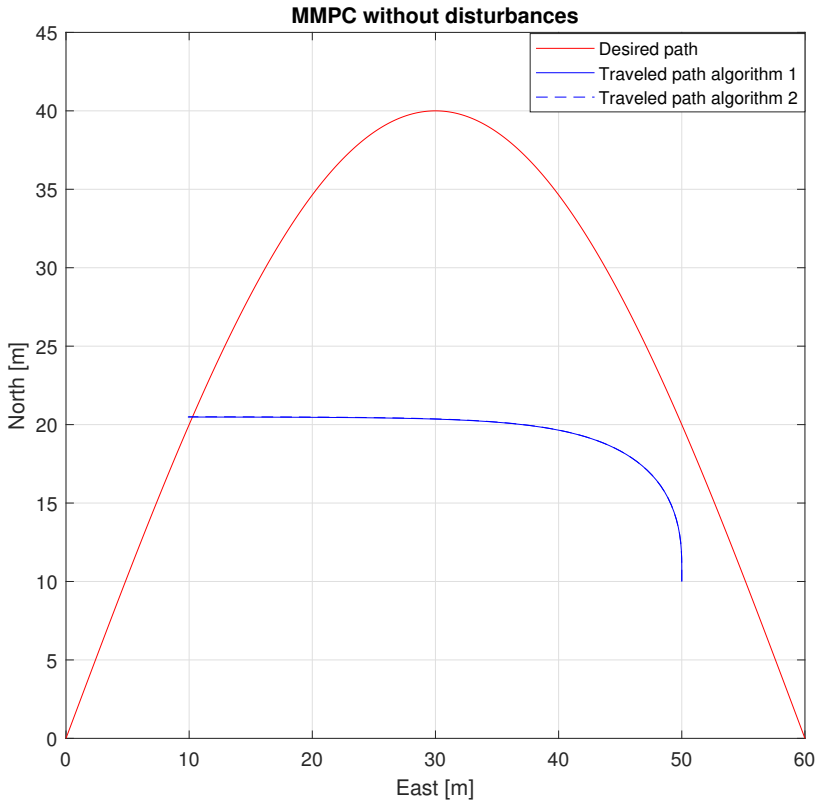

Figure 6.39: Multiplexed Model Predictive Control without disturbances with $N = 40$

The MMPC was further tested for $\mathbf{\Lambda}_1$ for initial sates $[x^e(0),\ \ y^e(0),\ \ \psi(0),\ \ r(0)]^T = [55,\ \ 35,\ \ \frac{\pi}{2}, 0]^T$ with both algorithms without disturbances. The MMPC was simulated with prediction horizons $N = 10$, $N = 20$, $N = 30$ and $N = 40$, and all simulations gave the results shown in **Fig. 6.40**

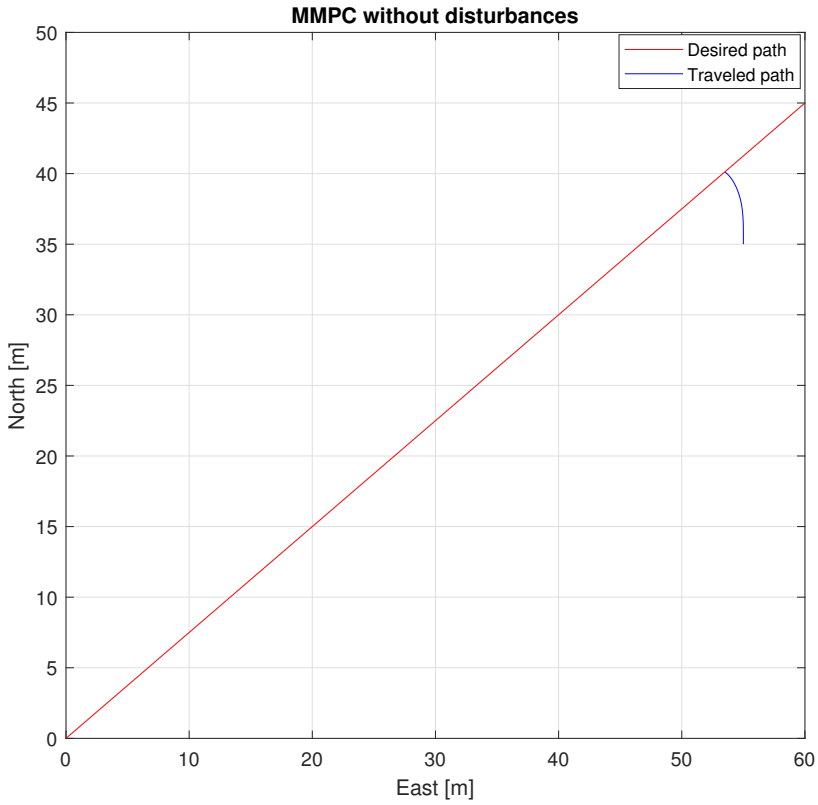

Figure 6.40: Multiplexed Model Predictive Control without disturbances

6.6 Run time

As an important aspect of the MPC is the run time of the algorithm, we will now compare the run time for the different methods. The computation times for the 3-DOF NMPC with APMonitor's Active Set method APOPT and IP method IPOPT are shown in **Table 6.1**.

N	Run time APOPT [s]	Run time IPOPT [s]
10	0.062	0.2
20	0.22	0.42
25	0.37	0.62
30	0.65	0.87
40	1.2	2.59

Table 6.1: Computation times for APM solvers for 3-DOF NMPC

In addition to these computation times, the simulations took an additional 1.72s on average to connect to the online server, which must be added to the total run time for the algorithm.

The run time for the MMPC control law with the APMonitor's APOPT solver is shown in **Table 6.2**

N	Run time outer loop [s]	Run time inner loop [s]
10	0.079	0.017
20	0.36	0.023
30	0.59	0.032
40	0.88	0.044

Table 6.2: Computation times for APM solvers for 3-DOF NMPC

For the Python simulations using the open source SciPy's SQP solver, the run time of the 3-DOF NMPC with different prediction horizons are shown in **Table 6.3**.

N	Run time SciPy's SQP solver [s]
10	1.36
20	6.87
30	36.81
40	116.32

Table 6.3: Computation times for SciPy SQP solver for 3-DOF NMPC

For the Python simulations of the MMPC, using SciPy's SQP solver, the outer and inner loop run times with different prediction horizons are shown in **Table 6.4**.

N	Run time outer loop MPC [s]	Run time inner loop MPC [s]
10	1.42	0.059
20	8.73	0.18
30	12.51	0.33
40	48.9	0.61

Table 6.4: Run times for SciPy's SQP solver for 3-DOF MMPC

A comparison of the run times of the simplified 6-DOF and the 3-DOF MPC showed that for $N = 40$ the simplified 6-DOF controller had a 0.2 seconds larger run time on average. This comparison was done by saving the run time for each iteration for the entire simulation for both controllers, before averaging the results. It is worth noting here that these average run times also include connection times to the server.

An attempt was also made to implement the MMPC control law on OLAV. This was done by making two python scripts, one for the inner and one for the outer loop, using algorithm 2. OLAV is controlled using Robotic Operating System (ROS) running on Ubuntu 16.04. Therefore, in order to implement the MMPC, a VM was set up with Ubuntu 16.04 and ROS, and the necessary software for OLAV's low level controllers was installed. Further the python scripts realizing the MMPC was implemented in OLAV's control system, and an attempt was made to test the control law for $\mathbf{\Lambda}_1$ with $N = 10$.

This implementation had a run time for the outer loop of 1.92 seconds, and a run time of 0.38 seconds for the inner loop. As OLAV requires inputs at a sampling rate of 0.1 seconds, OLAV was forced into an emergency stop. There are therefore no results to show for the implementation other than these run times.

Chapter 7

Discussion

7.1 Control law performance

When looking at the performance of the linearized MPC, we can see that this control law is unable to follow the path as desired. This is despite the fact that the linearized system was shown to be controllable in **Section 4.3**. The reason for this might be that stability can not be guaranteed when using a MPC as stated in Foss and Heirung (2016) and Grune and Pannek (2017). Further, one problem with this controller might be the uncertainties caused by linearization, e.g that the system is always linearized for $u > 0.1$. When examining the control inputs for the linearized controllers, we can see that for the end of the simulation, the steering angle $\gamma = 35^\circ$ whilst the velocity is $u = 0$. As this implies that the system is linearized about $u = 0.1$, the linear system has the control matrix

$$B = \begin{bmatrix} \cos\psi^* & 0 \\ \sin\psi^* & 0 \\ 0 & 0 \\ \frac{2u^* R\mu_f \sin\gamma^*}{I_z} & \frac{u^*|u^*|R\mu_f \cos\gamma}{I_z} \end{bmatrix} = \begin{bmatrix} \cos\psi^* & 0 \\ \sin\psi^* & 0 \\ 0 & 0 \\ \frac{1.26 \cdot 37.9 \cdot 2 \cdot 0.1 \cdot \sin 35^\circ}{1074} & \frac{1.26 \cdot 37.9 \cdot 0.01 \cos 35^\circ}{1074} \end{bmatrix} \tag{7.1}$$

As we can see, the element at $B_{4,2}$ is non zero, which implies that the control input γ will affect the system. However in reality, this is not the case as the vehicle can not turn without a velocity, regardless of the steering angle γ. Due to the linearization, the MPC is modelled such that it should be able to control the yaw rate without any velocity, which is not the case in reality. The linearized control law as it is formulated in **Section 4.3**, is unable to compensate for this. This is one of the weakness' of the MPC approach as opposed to e.g a Proportional Integral Derivative (PID)-controller, which would gradually increase u while the error persists and γ is saturated, and thus achieve convergence.

When comparing the results from **Section 6.3**, we can see that the NMPC for 3-DOF with both the IP and the Active Set solver, is able to follow both the linear and curved path, i.e $\mathbf{\Lambda}_1$ and $\mathbf{\Lambda}_2$, as long as the prediction horizon is chosen sufficiently large. The length of the prediction horizon has an important function as it enables the vehicle to move further

away from the path, as long as this will bring the vehicle closer to the final waypoint for future time steps. E.g by allowing the UGV to drive further away from the path direction in order to turn the vehicle. If we consider this example with k_1 equal to the number of time steps required to turn the vehicle, the objective function will increase for every time step $k < k_1$, however it will decrease for $k_1 < k$. Thus, if the number of time steps after the vehicle has turned, i.e $N - k_1$ is large enough, the total objective function value will be decrease by turning the vehicle, as the penalty for deviations are multiplied for every time step of the prediction horizon in the objective function. As long as the reduced objective function value for $k = k_1 \rightarrow N$ outweighs the increased objective function value for $k = 1 \rightarrow k_1$, the optimal solution of the objective function will be to turn the UGV. such that it is able to converge towards the path.

One important thing to note from the simulations, is that for some initial conditions, the MPC seems to perform better with disturbances than without. One possible cause for this is that for some situations, the disturbances might serve as a damping, i.e damping the turning more than what is modelled. E.g if $D = 1.2D_0$ whilst $\mu_f = 0.8\mu_{f0}$, this will decrease the UGV's turning rate faster than the ideal model with the same inputs, and thus reduce the distance the UGV overshoots while turning.

The simulations also show that for some initial states with disturbances, the 3-DOF controller performed better than the simplified 6-DOF controller. This is caused by the coefficients δ_D and δ_μ giving a lower damping change of turning rate, which will counteract the effect of the inclination. Therefore, as the simplified 6-DOF controller compensates for the kinematics, it will cause the simulated UGV to overshoot. However, as the 3-DOF controller does not compensate for the kinematics, it will converge to the path faster. This can be seen in **Fig. 6.29**, where the 6-DOF controller converges faster to the path than the 3-DOF controller, compared to **Fig. 6.32**, where the 6-DOF controller overshoots whilst the 3-DOF controller does not. The figures shows simulation with and without coefficient errors. For comparison, **Fig. 6.33** shows the same simulation where the coefficient errors are $\delta_D \in [1,\ \ 1.2]$ and $\delta_\mu \in [1,\ \ 1.2]$, and we can see that the 6-DOF controller has faster convergence than the 3-DOF controller, as this effect does not occur for these coefficient errors. For simulations without other disturbances than the kinematics, as seen in **Fig. 6.29**, we can see that the simplified 6-DOF control law has better performance, which is as expected as this control law compensates for the orientation in the terrain.

By comparing **Fig. 6.13** and **Fig. 6.24** we can see that the APMonitor's Active Set solver is able to follow the path $\mathbf{\Lambda}_2$. However, an implementation of the same control law using SciPy's SQP solver is unable to follow both paths with the same initial states. The same applies for the linear path, as shown in **Fig. 6.28**. This may imply that the SciPy SQP solver only finds suboptimal solutions to the optimization problem, which further implies that the inputs are not the optimal inputs to follow the path.

From **Fig. 6.35**, we can see that algorithm 1 performs better than algorithm 2 for the MMPC control law. For algorithm 1, the UGV oscillates around the desired path, whilst for algorithm 2, the UGV does not oscillate, however it takes a shortcut from the path. One possible explanation for this might be that for algorithm 1, the the outer loop does not account for the inner loop dynamics for each optimization, which will cause the vehicle to overshoot before the desired heading is reached. As algorithm 1 attempts to compensate for this for the next iteration, it will cause the vehicle to oscillate around the path due to

these overshoots. However for algorithm 2, the inner loop is able to reach the desired heading within each outer loop sampling time. The UGV thus follows the piece wise linear path from the outer loop MPC, however due to the inner loop dynamics, the UGV will not reach the predicted position for each outer loop iteration. As the outer loop inputs calculates an input for a much larger sampling time than for algorithm 1, the control law will attempt to converge to a point further along the path, in order to avoid overshoots.

FFI are currently using a feedback based control law on the UGV, which is based on the control law described in Siciliano and Khatib (2008). This control law aims to maintain a nominal velocity and steering to follow a prerecorded trajectory. Compensations in the control inputs are added to these nominal inputs to compensate for deviations from the trajectory. Both the existing control law, and the MPC control law derived in this thesis, have different advantages compared to each other. As the existing controller is feedback based, it has the advantage that stability can be guaranteed at all times. The MPC control law can not guarantee stability at at all times, as there does not exist a stability proof for MPCs. The MPC however offers an advantage as it does not require a prerecorded path nor nominal inputs. This is advantageous for implementations on ISR-missions, as the paths can be defined as a continuous function of s, and does not require the path to be prerecorded by driving it once. Further, the MPC does not require known velocity and steering angle inputs in order to follow the path, as these will be found by solving the optimization function.

As well as calculating the optimal input to reach the desired velocity, the MPC also calculates the optimal desired position on the path at each time step. This is advantageous for some implementations, as we do not require the UGV to follow the path from the beginning, but rather converging to the closest position on the path before following the path towards the end. This is preferred when it is desirable to reach the path and the end point as soon as possible e.g when traveling between two points. However, there are some implementations where this is not desired, e.g when OLAV is used for mapping purposes. If one considers a case where OLAV is used to map an area, it might be desired to follow the entire path from beginning to end, such that the entire area is mapped. For implementations where this is desired, the objective formulation from Ling et al. (2011) might be desired. As Ling et al. (2011) defines s such that $\dot{s} = v$, the controller can be initialized at $s = 0$, before $v > 0$ ensures that the UGV will progress along the path once the beginning of the path is reached. It is important to note here that Ling et al. (2011) defines $s \to n$ as progress in the path direction, whilst we for this thesis have defined $s \to 0$ as progress in the path direction.

One possible modification to the control law might be to enforce $s = n$ for the initial position, which will cause the UGV to converge towards the beginning of the path. This could be enforced until the UGV is within an accepted distance from the initial path position, before allowing s to be chosen freely. This might ensure that the UGV follows the path from the beginning to the end, for implementations were this is desired.

Another path implementation where the control law derived in this thesis fails, is when the desired path forms a closed cycle, e.g the eight-shaped path from Yu et al. (2011). As the objective function aims to minimize s to reach the end point, the MPC will stop when $\Lambda(0)$ is reached rather than repeating the path for another cycle. This is the main difference between the MPC formulation derived in this thesis and the one derived in Ling et al.

(2011). As Ling et al. (2011) defines s such that $\dot{s} = v$, $v > 0$ will continually increase s such that the vehicle advances along the closed path cycle. The choice of s as a parameter that can be chosen freely for each iteration as defined in this thesis however, offers an advantage when attempting to reach a desired end point on the path, as described earlier. By choosing s freely for each iteration, we also reduce the number of optimization variables as we remove v from the optimization formulation. This is an advantage when it comes to the computation time, as the reduced number of variables improves the computation time of the optimization.

One modification that could be made to allow the UGV to follow closed cycle paths, might be to attempt to keep the vehicle on the path whilst the velocity is kept constant. This could be achieved by minimizing $(u - u_d)^2$ where u_d is some nominal velocity, rather than minimizing s. The MPC might thus attempt to maintain a constant velocity, whilst simultaneously attempting to keep the UGV on the path. As this modification of the control law aims to maintain a constant velocity instead of reaching the final waypoint, it might be able to follow the cycle paths.

7.2 Computation time

When comparing the computation times from **Table 6.1** to **Table 6.4** we can see that the run time increases when the prediction horizon is increased. This is as expected, as Ling et al. (2011) describes that the run time is given as $O((m \times N)^3)$. Further, we can see that for both solvers, the run time is generally lower for the MMPC approach than for the NMPC approach, which is also as expected due to the reduced number of states.

One important thing to note is that by comparing **Table 6.3** and **Table 6.4**, we can see that the computation time for $N = 20$ is lower for the NMPC than the MMPC. The same can be seen for $N = 10$ for the 3-DOF NMPC and the MMPC with the APOPT solver. This is the opposite of what is expected as the MMPC has a reduced number of states for each optimization, and as the MMPC shows reduced computation time compared to the NMPC for all other values of N. The results from Ling et al. (2011) also showed faster computation time for the regular MPC approach for smaller prediction horizons. For the tests done in this thesis however, these were the only cases that showed faster computation time for the regular NMPC approach than for the MMPC approach. The simulations were run several times, and the computation times were consistent, which implies that this is not an error of measurement. One possible explanation might be that the Active Set solver is able to eliminate some infeasible solutions that violate the constraints of the yaw dynamic for the NMPC approach. However, as the outer loop MMPC does not consider the yaw dynamics, the Active Set solver is not able to eliminate these infeasible solutions, and will therefore converge slower towards the optimal solution.

The MMPC approach was chosen for the implementation on OLAV, as this approach gave a reduced computation time for the simulations. Further, algorithm 2 was chosen, as this allows for a longer outer loop run time than algorithm 1. However, as the computation times were still too large, the implementation was unable to control OLAV. As we can see from the results, the optimization had a run time of 1.92 seconds for the outer loop during the testing, whilst the sampling time of this controller was set to 1 seconds. This would have implied that any input from the outer loop MPC, i.e desired heading and velocity,

would have been applied to the system 0.92 seconds longer than intended. If we consider a vehicle driving at the maximum velocity of $5\mathrm{m/s}$, this might cause the UGV to overshoot by $4.5\mathrm{m}$ before a new control input is applied.

Further, the implementation was also affected by the large computation times of the inner loop MPC, as the optimization for the inner loop MPC took 0.38 seconds during the testing of the control law. This is four times larger than the specified sampling time for this controller. For the inner loop MPC, the delayed update of the input would have caused the UGV to turn past the desired heading angle, which would have further created oscillations around the desired heading angle, and thus an unstable system. However, as the implementation was unable to update the inputs within the sampling time, OLAV was forced into an emergency stop.

It is important to note here that for the implementation on OLAV, OLAV was controlled using a VM, which has a limited processing power compared to the machine used for the simulations. The simulations was ran on a fourth generation i7 processor with eight cores, whilst the VM was limited to four cores of this processor. This might significantly affect the computation time of the algorithm, and thus had an effect on the performance of the controller.

One solution to the large computation time, might be to reduce the maximum velocity, as this reduces the distance the UGV is able to travel in one time step. Thus, by extension, this will reduce the distance the UGV overshoots before a new control input is applied. Further, the sampling time of the controller can be increased, e.g by letting the outer loop MPC calculate an input for $2\mathrm{s}$ instead of $1\mathrm{s}$, as this would allow the controller to be able to find a new optimal input within the specified sampling time.

Another solution to the problem might be to apply the next input in the control sequence if no solution is found within the time interval, as proposed by Zhang et al. (2015). As the kinematic model only considers an objects motion in a space without the cause of said motion, the kinematic model will not be affected by any disturbances as long as the UGV is able to maintain the desired velocity and heading angle. For a kinematic model, the only disturbances are the unmodelled kinematics, e.g by using a kinematic model for 3-DOF instead of 6-DOF, which is inaccurate if there are large inclinations of the terrain. Further, this approach might be more accurate for a UGV application, as there are less external disturbances affecting the system, than for many other control systems, e.g ocean vessels, which is affected by currents and waves, as this would imply that $v \neq 0$ in (3.1). The limitation with this approach however, is that the UGV can not change the heading angle momentarily, and thus we have $\psi_d \neq \psi$ for much of the sampling time. As the outer loop MPC does not consider the vehicle's motion before the desired heading is reached and how this affects the position, this would cause an inaccuracy in the position that is not compensated for by the next input in the input sequence.

The solution of applying the second input of the input sequence might however be less effective for the inner loop than for the outer loop, as the model is based on the dynamics and are more prone to disturbances. These disturbances will mostly consist of modelling errors, as parameters such as the damping and friction coefficients will vary depending on the surface. Any deviations from the modelled parameters might cause inaccuracies in the optimal input, which might increase for later time steps of the input sequence.

A simplification of the heading dynamics might be included in the outer loop MPC

in order to compensate for some of the inaccuracies. This could be done by including a maximum change of heading angle for each iteration, or by also minimizing the change of heading. This will however increase the number of states in the optimization, which might further increase the computation time.

It is also worth noting here that as the measurements of the current states are taken at the beginning of the algorithm, these will vary from the actual states when the input is applied. This effect will be greater for greater run times, as more time has passed were the states have been affected by the previous inputs. Ideally the optimization should be able to run within the minimal sampling time of the states, as this would reduce this effect to a minimum. For the simulations however, the optimization is run after the states are measured, and the system is not simulated for the next time step before the new input is applied. This might cause unrealistically good results in the simulations compared to the actual implementation, as this implies that there is no time between the measurements and the inputs are applied.

When comparing the run time of the MPC control laws derived in this thesis and FFI's existing control law, FFI's control law has a significant advantage. This is because FFI's feedback based control law only requires mathematical operations such as multiplication and trigonometric functions, and does not require an optimization problem to be solved at each iteration. Thus, the existing control law has an insignificant run time, and is able to generate new inputs within the sampling time, which was not possible for the solvers tested for the MPC control law.

Chapter 8

Conclusion and future work

8.1 Conclusion

When looking at the results from **Section 6.1**, we can see that the simulated object converges to the path and follows this to the end point. We can thus conclude that the basic optimization formulation that was derived for path following problems in this thesis works as desired for ideal systems.

The simulations of the UGV with the linearized MPC shows that the linearized model was too inaccurate to be able to follow the path in simulations. The linearized MPC had the shortest run time of of controllers, which is expected as QP solvers can be used. However, as the linearized MPC does not follow the path as desired, it is not a candidate for implementation regardless of the low run time of the algorithm.

Further, the simulations with the modelled vehicle dynamics from **Section 6.3** show that the simulated UGV is able to follow the path with the 3-DOF NMPC. This control law was able to follow the path, both with and without disturbances for all initial conditions, given that the prediction horizon was chosen long enough. By comparing the different solvers for this controller, we can see that there is no difference in performance between the APMonitor's IP solver IPOPT and Active Set solver APOPT. However, when comparing the run times, we see that the Active Set methods have a faster run time, which makes it more desirable for implementation as the run time is an issue.

We can also see from **Section 6.4** that the simplified 6-DOF control law performs better than the 3-DOF control law as the kinematics are compensated for. Both controllers are able to follow the path with large pitch angles and disturbances, however the simplified 6-DOF control law converges to the path faster than the 3-DOF control law. When comparing the average run time for the two control laws, there is no significant increase in run time for the simplified 6-DOF controller, and we can conclude that the simplified 6-DOF NMPC is a better choice of control law.

From **Section 6.3.2**, we can see that SciPy's SQP solver was unable to follow the path for any choice of prediction horizon for the 3-DOF NMPC. We can thus conclude that the SciPy SQP solver finds suboptimal solutions to the optimization problem compared to the

APMonitor solvers, and is thus not suited for implementation of this control law.

Neither the APMonitor solvers nor the SciPy SQP solver was able to follow the path with the MMPC control law, which is likely caused by the fact that this formulation is too inaccurate, as the outer loop MPC does not account for the change of ψ. The MMPC approach did however have lower run times, and we can conclude that the MMPC approach is faster, but is too inaccurate to control the UGV.

The MPC control law perform well in simulations as the UGV converges to the path and follows the path to the end point for all initial states despite disturbances. and show potential for implementation. We can thus conclude that the MPC path following control law shows potential for implementation provided one can find a solver that is able to solve the optimization within the sampling time of the UGV.

8.2 Future work

As the main set back of the control laws of this thesis has been the run time and implementation, our recommendation for future work would be to find efficient solvers for the optimization problem. This thesis has been limited to open source solvers for MATLAB and Python, however there exists a variety of solvers for C++ implementation, as well as commercial solvers that have not been considered. Some of these solvers might be more efficient than the solvers considered by this thesis, and would therefore be relevant to look into.

Bibliography

Auby, P. M., 2016. Modeling and parameter estimation of an unmanned ground vehicle with a continuously variable transmission. Master thesis 1 (1), 1–100.

Copp, D. A., Hespanha, J. P., 2016. Simulteneous nonlinear model predictive control and state estimation. Automatica 77 (1), 1–12.

Dyrnes, S. B., 2018. Modelling of unmanned ground vehicle in 6 degrees of freedom for implementation of model predictive control. Pre project 1 (1), 1–40.

Foss, B., Heirung, T. A., 2016. Merging optimization and control. Technical report 1 (1), 1–80.

Fossen, T., 2011. Handbook of Marine Craft Hydrodynamics and Motion Control. John Wiley & Sons.

Grune, L., Pannek, J., 2017. Nonlinear Model Predicitive Control. Springer.

Ling, K. V., Maciejowski, J., Richards, A., Wu, B. F., 2011. Multiplexed model predicitve control. Automatica 48 (2), 396–401.

Mashadi, B., Ahmadizadeh, P., Majidi, M., Mahmoodi-Kaleybar, M., 2014. Integrated robust controller for vehicle path following. Multibody System Dynamics 33 (2), 207–228.

Mata, S., Zubizaretta, A., Cabanes, I., Nieva, I., Pinto, C., 2017. Linear time varying model predictive control for lateral path tracking. Int. J. Vehicle Design 75 (1), 1–22.

Mathiassen, K., Baksaas, M., Olsen, L. E., Thoresen, M., Tveit, B., 2016. Development of an autonomous off-road vehicle for surveillance missions. NATO Journal 1 (1), 1–8.

Miller, R. E., 2011. Optimization: Foundations and applications. John Wiley & Sons.

Nocedal, J., Wright, S. J., 2006. Numerical Optimization. Springer New York.

Siciliano, B., Khatib, O., 2008. Handbook of robotics. Springer.

Yu, S., Li, X., Chen, H., Allgower, F., 2011. Nonlinear model predictive control for path following problems. Automatica 25 (8), 1169–1189.

Zhang, K., Sprinkle, J., Sanfelice, R. G., 2015. A hybrid model predictive controller for path planning and path following. In: Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems. Seattle, Washington.

Appendix

Matlab code for NMPC 3-DOF control law

```
%% System parameters
h = 0.1; % Sampling time
I_z = 1074; %Moment of inertia
D = 1528; %Damping coefficient
mu_f = 37.9; %Friction coefficient
R = 1.26; %Distance from centre of vehicle to front wheels

%% Define path
s = 0:0.01:30;
lambda_x = 2*s;
lambda_y = 40*sin(s*pi/30);

%% Defines empty vectors for the states, input and slack variables
x = NaN(351,4);
u = NaN(350,2);
e = NaN(350,2);
x(1,:) = [50, 10, pi/2, 0]; %Initial position
%% Writes the initial position to a csv fila that is uploaded to the op
dlmwrite('initial.csv','NEPR');
dlmwrite('initial.csv',x(1,:),'-append');

s = NaN(1,250);
for t = 1:350
    tic; %Start time
    addpath('../apm');

    % Select server
    server = 'http://byu.apmonitor.com';

    % Application name
    app = 'trial';

    % Clear previous application
    apm(server,app,'clear all');

    % Load model file
    apm_load(server,app,'nmpc.apm');
```

```
    % load data
    csv_load(server,app,'initial.csv');

    % Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
    apm_option(server,app,'nlc.solver',3);

    % Solve on APM server
    apm(server,app,'solve')
    results = apm_sol(server,app)
    values = cell2mat(results(2:end,:));

    u(t,1) = results.values(169); %Get velocity input
    u(t,2) = results.values(209); %Get steering angle input
    T(t) = toc; %End time, T(t) will be the computation time of the solv
    %% Simulate system for one time step
    ct = cos(0.34*rand()); %Random number simulating change of pitch ang
    x(t+1,1) = x(t,1) + h*ct(t,1)*u(t,1)*cos(x(t,3));
    x(t+1,2) = x(t,2) + h*ct(t,1)*u(t,1)*sin(x(t,3));
    x(t+1,3) = x(t,3) + h*ct(t,1)*x(t,4);
    x(t+1,4) = x(t,4)-h*D*x(t,4)*abs(x(t,4))*delta_D(t,1)/I_z + h*R*mu_1
    %% Upload new posititons to csv file as initial positions
    dlmwrite('initial.csv','NEPR');
    dlmwrite('initial.csv',x(t+1,:),'-append');
end

%% Plot the desired and traveled path in figure 1

figure(1)
plot(lambda_x, lambda_y, 'r')
hold on
plot(x(:,1),x(:,2),'b')
hold off
```

Code for APmonitor solver NMPC 3-DOF

```
Model
Parameters
        N ! initial x^e
        E ! initial y^e
        P ! initial psi
        R ! initial r
End Parameters
Constants
  nx = 2
  nu = 2
  k = 40 ! prediction horizon
  h = 0.1 ! sampling time
```

```
End Constants
  Variables
    x[1:nx*k] = 1, >=-1000, <=1000
        x[k*nx+1:k*(nx+1)] = 3.14, >=-2*3.14, <= 4*3.14
        x[k*(nx+1)+1:k*2*nx] = 0, >=-1.77, <= 1.77
    u[1:k] = 1, >=0, <=5
        u[k+1:k*nu] = 0, >=-35*3.14/180, <= 35*3.14/180
        s[1:k] = 0, >=0, <= 30
        epsilon[1:k*nx] = 1, >=-1000, <= 1000
  End Variables

  Equations
        ! Constraints for x^e
    x[1:k] + epsilon[1:k] = 2*s[1:k] ! Path constraint
        x[1] - h*u[1]*cos(P) = N ! Dynamic constraint for first time ste
        x[2:k] - x[1:k-1] - h*u[2:k]*cos(x[k*nx+1:k*(nx+1)-1]) = 0 ! Dyn

        ! Constraints for y^e
        x[k+1:k*nx] + epsilon[k+1:k*nx] =  40*sin((s[1:k])*3.14/30) ! Pa
        x[k+1] - h*u[1]*sin(P) = E ! Dynamic constraint for first time s
        x[k+2:k*nx] - x[k+1:k*nx-1] - h*u[2:k]*sin(x[k*nx+1:k*(nx+1)-1])

        ! Constraints for psi
        x[k*nx+1] = P+h*R ! Dynamic constraint for first time step
        x[k*nx+2:k*(nx+1)] - x[k*nx+1:k*(nx+1)-1] - h*x[k*(nx+1)+1:k*2*n

        ! Constraints for r
        x[k*(nx+1)+1] - h*37.9*1.26*u[1]*abs(u[1])*sin(u[k+1])/1074 = R-
        x[k*(nx+1)+2:k*2*nx] - x[k*(nx+1)+1:k*2*nx-1]+1528*h*x[k*(nx+1)+

        ! Minimization function
    minimize  (s[1:k])^2+1000*(epsilon[1:k])^2+1000*(epsilon[k+1:k*nx])^
  End Equations
End Model
```

Matlab code for simplified 6-DOF NMPC control law

```
%% System parameters
h = 0.1; % Sampling time
I_z = 1074; %Moment of inertia
D = 1528; %Damping coefficient
mu_f = 37.9; %Friction coefficient
R = 1.26; %Distance from centre of vehicle to front wheels

%% Define path
s = 0:0.01:30;
lambda_x = 2*s;
```

```
lambda_y = 40*sin(s*pi/30);

%% Defines empty vectors for the states, input and slack variables
x = NaN(351,4);
u = NaN(350,2);
e = NaN(350,2);
x(1,:) = [50, 10, pi/2, 0]; %Initial position
%% Uses random variables to determine disturbances, and set a constant p
ct = cos(30*pi/180*ones(350,1));
cf = cos(30*pi/180*rand(350,1));
delta_D = (1.2-0.4*rand(350,1));
delta_mu = (1.2-0.4*rand(350,1));
for t = 1:350
    %% Writes the initial position and orientation to a csv fila that is
    dlmwrite('initial.csv','NEPRcf');
    dlmwrite('initial.csv',[x(t,:), ct(t,1), cf(t,1)],'-append');
    tic; %Start time
    addpath('../apm');

    % Select server
    server = 'http://byu.apmonitor.com';

    % Application name
    app = 'trial';

    % Clear previous application
    apm(server,app,'clear all');

    % Load model file
    apm_load(server,app,'en_nmpc.apm');

    % load data
    csv_load(server,app,'initial.csv');

    % Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
    apm_option(server,app,'nlc.solver',3);

    % Solve on APM server
    apm(server,app,'solve');
    results = apm_sol(server,app)
    values = cell2mat(results(2:end,:));
    u(t,1) = results.values(171); %Get velocity input
    u(t,2) = results.values(211); %Get steering angle input
    T2(t) = toc; %End time, T(t) will be the computation time of the sol
    %% Simulate system for one time step
```

```
    x(t+1,1) = x(t,1) + h*ct(t,1)*u(t,1)*cos(x(t,3));
    x(t+1,2) = x(t,2) + h*ct(t,1)*u(t,1)*sin(x(t,3));
    x(t+1,3) = x(t,3) + h*x(t,4)*cf(t,1)/ct(t,1);
    x(t+1,4) = x(t,4)-h*D*x(t,4)*abs(x(t,4))*delta_D(t,1)/I_z + h*R*mu_f
    e(t,1) = results.values(291);
    e(t,2) = results.values(331);
end

%% Plot the desired and traveled path in figure 1

figure(1)
plot(lambda_x, lambda_y, 'r')
hold on
plot(x(:,1),x(:,2),'b')
hold off
```

Matlab code for MMPC control law algorithm 1

```
%% System parameters
h = 0.1; % Sampling time
I_z = 1074; %Moment of inertia
D = 1528; %Damping coefficient
mu_f = 37.9; %Friction coefficient
R = 1.26; %Distance from centre of vehicle to front wheels

%% Define path
s = 0:0.01:30;
lambda_x = 2*s;
lambda_y = 40*sin(s*pi/30);

%% Defines empty vectors for the states, input and slack variables
x = NaN(351,4);
u = NaN(350,2);
e = NaN(350,2);
x(1,:) = [50, 10, pi/2, 0]; %Initial position
%% Writes the initial position to a csv fila that is uploaded to the op
dlmwrite('initial_o.csv','NE');
dlmwrite('initial_o.csv',x(1,1:2),'-append');
for t = 1:250
    %% Outer loop
    tic;
    addpath('../apm');

    % Select server
    server = 'http://byu.apmonitor.com';

    % Application name
```

```
app = 'trial';

% Clear previous application
apm(server,app,'clear all');

% Load model file
apm_load(server,app,'mmpc_o.apm');

% load data
csv_load(server,app,'initial_o.csv');

% Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
apm_option(server,app,'nlc.solver',3);

% Solve on APM server
apm(server,app,'solve')
results = apm_sol(server,app)
values = cell2mat(results(2:end,:));
u(t,1) = results.values(87); %Get velocity inpu
u(t,2) = results.values(127); %Get desired heading
e(t,1) = results.values(207);
e(t,2) = results.values(247);
T(t) = toc; %End time, T(t) will be the computation time of the oute

%% Inner loop
tic;
% Writes the initial orientation, velocity and desired heading to a
dlmwrite('initial_i.csv','PRuD');
dlmwrite('initial_i.csv',[x(t,3:4), u(t,:)],'-append');
addpath('../apm');

    % Select server
server = 'http://byu.apmonitor.com';

    % Application name
app = 'trial';

    % Clear previous application
apm(server,app,'clear all');

    % Load model file
apm_load(server,app,'mmpc_i.apm');

    % load data
csv_load(server,app,'initial_i.csv');
```

```
        % Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
    apm_option(server,app,'nlc.solver',3);

    % Solve on APM server
    apm(server,app,'solve');
    results = apm_sol(server,app)
    values = cell2mat(results(2:end,:));
    T2(t) = toc; %End time, T(t) will be the computation time of the inn
    gamma(t,1) = results.values(88); %Get the steering angle input
    %% Simulate system for one time step
    x(t+1,1) = x(t,1) + h*u(t,1)*cos(x(t,3));
    x(t+1,2) = x(t,2) + h*u(t,1)*sin(x(t,3));
    x(t+1,3) = x(t,3) + h*x(t,4);
    x(t+1,4) = x(t,4)-h*D*x(t,4)*abs(x(t,4))/I_z + h*R*mu_f*u(t,1)*abs(u
    %% Writes the initial position to a csv fila that is uploaded to the
    dlmwrite('initial_o.csv','NE');
    dlmwrite('initial_o.csv',x(t,1:2),'-append');
end

%% Plot the desired and traveled path in figure 1

figure(1)
plot(lambda_x, lambda_y, 'r')
hold on
plot(x(:,1),x(:,2),'b')
hold off
```

Matlab code for MMPC control law algorithm 2

```
%% System parameters
h = 0.1; % Sampling time
I_z = 1074; %Moment of inertia
D = 1528; %Damping coefficient
mu_f = 37.9; %Friction coefficient
R = 1.26; %Distance from centre of vehicle to front wheels

%% Define path
s = 0:0.01:30;
lambda_x = 2*s;
lambda_y = 40*sin(s*pi/30);

%% Defines empty vectors for the states, input and slack variables
x = NaN(351,4);
u = NaN(350,2);
e = NaN(350,2);
x(1,:) = [50, 10, pi/2, 0]; %Initial position
```

```
%% Writes the initial position to a csv fila that is uploaded to the opt
dlmwrite('initial_o.csv','NE');
dlmwrite('initial_o.csv',x(1,1:2),'-append');
for t = 1:250
    %% Outer loop
    tic;
    addpath('../apm');

    % Select server
    server = 'http://byu.apmonitor.com';

    % Application name
    app = 'trial';

    % Clear previous application
    apm(server,app,'clear all');

    % Load model file
    apm_load(server,app,'mmpc_o.apm');

    % load data
    csv_load(server,app,'initial_o.csv');

    % Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
    apm_option(server,app,'nlc.solver',3);

    % Solve on APM server
    apm(server,app,'solve')
    results = apm_sol(server,app)
    values = cell2mat(results(2:end,:));
    u(t,1) = results.values(87); %Get velocity inpu
    u(t,2) = results.values(127); %Get desired heading
    e(t,1) = results.values(207);
    e(t,2) = results.values(247);
    T(t) = toc; %End time, T(t) will be the computation time of the solv

    %% Inner loop
    for i=1:10
        tic;
        % Writes the initial orientation, velocity and desired heading t
        dlmwrite('initial_i.csv','PRuD');
        dlmwrite('initial_i.csv',[x((t-1)*10+i,3:4), u(t,:)],'-append');
        addpath('../apm');

            % Select server
```

```
        server = 'http://byu.apmonitor.com';

            % Application name
        app = 'trial';

            % Clear previous application
        apm(server,app,'clear all');

            % Load model file
        apm_load(server,app,'mmpc_i.apm');

            % load data
        csv_load(server,app,'initial_i.csv');

            % Option to select solver (1=APOPT, 2=BPOPT, 3=IPOPT)
        apm_option(server,app,'nlc.solver',3);

        % Solve on APM server
        apm(server,app,'solve');
        results = apm_sol(server,app)
        values = cell2mat(results(2:end,:));
        T2(t) = toc; %End time, T(t) will be the computation time of the
        gamma((t-1)*10+i,1) = results.values(88); %Get the steering angl
        %% Simulate system for one time step
        x((t-1)*10+i+1,1) = x((t-1)*10+i,1) + h*u(t,1)*cos(x((t-1)*10+i,
        x((t-1)*10+i+1,2) = x((t-1)*10+i,2) + h*u(t,1)*sin(x((t-1)*10+i,
        x((t-1)*10+i+1,3) = x((t-1)*10+i,3) + h*x((t-1)*10+i,4);
        x((t-1)*10+i+1,4) = x((t-1)*10+i,4)-h*3300*x((t-1)*10+i,4)*abs(x
    end
    %% Writes the initial position to a csv fila that is uploaded to the
    dlmwrite('initial_o.csv','NE');
    dlmwrite('initial_o.csv',x(t,1:2),'-append');
end

%% Plot the desired and traveled path in figure 1

figure(1)
plot(lambda_x, lambda_y, 'r')
hold on
plot(x(:,1),x(:,2),'b')
hold off
```

Code for APmonitor solver MMPC outer loop

```
Model
Parameters
        N ! initial x^e
```

```
        E ! initial y^e
End Parameters
Constants
  nx = 2
  nu = 2
  k = 40 ! prediction horizon
  h = 0.1 ! sampling time
End Constants
  Variables
    x[1:nx*k] = 1, >=-1000, <=1000
    u[1:k] = 1, >=0, <=5
        u[k+1:k*nu] = 0, >=0, <= 2*3.14
        s[1:k] = 0, >=0, <= 30
        epsilon[1:k*nx] = 1, >=-1000, <= 1000
  End Variables

  Equations
    ! Constraints for x^e
    x[1:k] + epsilon[1:k] = 2*s[1:k] ! Path constraint
        x[1] - h*u[1]*cos(u[k+1]) = N ! Dynamic constraint for first tim
        x[2:k] - x[1:k-1] - h*u[2:k]*cos(u[k+2:k*nx]) = 0 ! Dynamic con

        ! Constraints for y^e
        x[k+1:k*nx] + epsilon[k+1:k*nx] =  40*sin((s[1:k])*3.14/30) ! Pa
        x[k+1] - h*u[1]*sin(u[k+1]) = E ! Dynamic constraint for first t
        x[k+2:k*nx] - x[k+1:k*nx-1] - h*u[2:k]*sin(u[k+2:k*nx]) = 0 ! Dy


        ! Minimization function
    minimize  (s[1:k])^2+1000*(epsilon[1:k])^2+1000*(epsilon[k+1:k*nx])^
  End Equations
End Model
```

Code for APmonitor solver MMPC inner loop

```
Model
Parameters
        P ! Initial heading
        R ! Initial yaw rate
        u ! Velocity
        D ! Desired heading
End Parameters
Constants
  nx = 2
  k = 40 ! prediction horizon
  h = 0.1 ! sampling time
End Constants
```

```
  Variables
        x[1:k] = 3.14, >=-2*3.14, <= 4*3.14 ! Heading
        x[k+1:k*nx] = 0, >=-1.77, <= 1.77 ! Yaw rate
        gamma[1:k] = 0, >=-35*3.14/180, <= 35*3.14/180 ! Steering angle
  End Variables

  Equations
        ! Constraints for psi
        x[1] = P+h*R ! Dynamic constraint for first time step
        x[2:k] - x[1:k-1] - h*x[k+1:k*nx-1] = 0 ! Dynamic constraint for

        ! Constraints for r
        x[k+1] - h*37.9*1.26*u*abs(u)*sin(gamma[1])/1074 = R-h*1528*R*ab
        x[k+2:k*nx] - x[k+1:k*nx-1]+1528*h*x[k+1:k*nx-1]*abs(x[k+1:k*nx-
        ! Minimization function
    minimize  1000*(D-x[1:k])^2+(gamma[1:k])^2+0.1*(x[k+1:k*nx])^2
  End Equations
End Model
```

Python code for 3-DOF NMPC control law

```
import numpy as np
from scipy.optimize import minimize
import time


global N, h, nx, init
init = [55.0, 10.0, np.pi, 0.0]
N = 40
h  = 0.1
nx = 9

"""Objective function"""
def obj(x):
    global N, h, nx
    s_obj = 0
    eps_x_obj = 0
    eps_y_obj = 0
    u1_obj = 0
    u2_obj = 0
    for k in range(N):
        s_obj = s_obj + x[4+k*nx]**2
        eps_x_obj = eps_x_obj + 1000*x[5+k*nx]**2
        eps_y_obj = eps_y_obj + 1000*x[6+k*nx]**2
        u1_obj = u1_obj + 0.1*x[7+k*nx]**2
        u2_obj = u1_obj + 0.1*x[8+k*nx]**2
```

```
    return s_obj+eps_x_obj+eps_y_obj+u1_obj+u2_obj

"""Constraint functions"""
def dc1(x):
    global h, init
    px_0 = init[0]
    return x[0]-h*x[7]*np.cos(x[2])-px_0

def dc2(x):
    global h, init
    py_0 = init[1]
    return x[1]-h*x[7]*np.sin(x[2])-py_0

def dc3(x):
    global h, init
    psi_0 = init[2]
    return x[2]-h*x[3]-psi_0

def dc4(x):
    global h, init
    r_0 = init[3]
    return x[3]-r_0 +h*1528*r_0*abs(r_0)/1074+h*1.26*37.9*x[7]*abs(x[7])

def dc_px(x):
    global N, h, nx
    px_cons = np.zeros(N-1)
    for k in range(1, N):
        px_cons[k-1] = x[k*nx] - x[(k-1)*nx] - h*x[7+k*nx]*np.cos(x[2+(k
    return px_cons

def dc_py(x):
    global N, h, nx
    py_cons = np.zeros(N-1)
    for k in range(1, N):
        py_cons[k-1] = x[1+k*nx] - x[1+(k-1)*nx] - h*x[7+k*nx]*np.sin(x[
    return py_cons

def dc_psi(x):
    global N, h, nx
    psi_cons = np.zeros(N-1)
    for k in  range(1, N):
        psi_cons[k-1] = x[2+k*nx] - x[2+(k-1)*nx] - h*x[3+(k-1)*nx]
    return psi_cons

def dc_r(x):
```

```
    global N, h, nx
    r_cons = np.zeros(N-1)
    for k in range(1, N):
        r_cons[k-1] = x[3+k*nx] - x[3+(k-1)*nx] +h*1528*x[3+(k-1)*nx]*ab
    return r_cons

def pc_px(x):
    global N, nx
    px_cons = np.zeros(N)
    for k in range(N):
        px_cons[k] = x[k*nx] - 2*x[4+k*nx] - x[5+k*nx]
    return px_cons

def pc_py(x):
    global N, nx
    py_cons = np.zeros(N)
    for k in range(N):
        py_cons[k] = x[1+k*nx] - 40*np.sin(x[4+k*nx]*np.pi/30) - x[6+k*r
    return py_cons


def simulate(x, y,  psi, r, u, gamma): #Simulate the system (euler metho
    global h
    D = 1528
    mu_f = 37.9
    R = 1.26
    x = x+h*u*np.cos(psi)
    y = y+h*u*np.sin(psi)
    psi = psi +h*r
    r = r - h*D*r*abs(r)/1074 + h*R*mu_f*u*abs(u)*np.sin(gamma)/1074
    return x, y, psi, r

def initial(x, y, psi, r): #Sets an initial search value for the first i
    global N, nx
    init = np.zeros(N*nx)
    for k in range(N):
        init[k*nx] = x
        init[1+k*nx]  = y
        init[2+k*nx] = psi
        init[3+k*nx] = r
        init[4+k*nx] = 15.0
        init[5+k*nx] = 0.0
        init[6+k*nx] = 0.0
        init[7+k*nx] = 5.0
        init[8+k*nx] = 0.0
```

```
    return init

def bounds(): #Defines inequality constraints
    global N
    bound = []
    for k in range(N):
        bound.append((-100.0, 100.0)) #bound for px at timestep k
        bound.append((-100.0, 100.0)) #bound for py at timestep k
        bound.append((-2*np.pi, 4*np.pi)) #bound for psi at timestep k
        bound.append((-np.pi/2, np.pi/2)) #bound for r at timestep k
        bound.append((0, 30)) #bound for s at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_x at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_y at timestep k
        bound.append((0.0, 5.0)) #bound for u at timestep k
        bound.append((-35*np.pi/180, 35*np.pi/180)) #bound for steering
    return bound


bnds = tuple(bounds())
"""Defines constraints in order"""
con1 = {'type': 'eq', 'fun': dc1}
con2 = {'type': 'eq', 'fun': dc2}
con3 = {'type': 'eq', 'fun': dc3}
con4 = {'type': 'eq', 'fun': dc4}
con5 = {'type': 'eq', 'fun': dc_px}
con6 = {'type': 'eq', 'fun': dc_py}
con7 = {'type': 'eq', 'fun': dc_psi}
con8 = {'type': 'eq', 'fun': dc_r}
con9 = {'type': 'eq', 'fun': pc_px}
con10 = {'type': 'eq', 'fun': pc_py}
cons = [con1, con2, con3, con4, con5, con6, con7, con8, con9, con10]

start=time.clock()
def main():
    global init
    tf = 350 #Set simulation time
    """Define empty arrays for the states, inputs and run time"""
    x = []
    y = []
    psi = []
    r = []
    runtime = []
    u = []
    gamma = []
    """Get initial states"""
```

```
    x0 = initial(init[0], init[1], init[2], init[3])
    x.append(init[0])
    y.append(init[1])
    psi.append(init[2])
    r.append(init[3])
    for i in range(1, tf):#Simulate system for the defined simulation ti
        start = time.clock()#Start timing of comptation time
        sol = minimize(obj,x0,method='SLSQP', bounds=bnds, constraints=c
        x0[0:(N-1)*nx] = sol.x[nx:N*nx+1] #Update inital search values
        x0[(N-1)*nx:N*nx+1] = sol.x[(N-1)*nx:N*nx+1] #Update inital sear
        runtime.append(time.clock()-start)#End timing of computation tim
        u.append(sol.x[7]) #Set velocity input
        gamma.append(sol.x[8]) #Set steering angle input
        x1, y1, psi1, r1 = simulate(x[i-1], y[i-1], psi[i-1], r[i-1], u[
        init = [x1, y1, psi1, r1] #Set initial state for next iteration
        x.append(x1) #Save x values
        y.append(y1) #Save y values
        psi.append(psi1) #Save psi values
        r.append(r1) #Save r values
    print(x)
    print(y)
    print(psi)
    print(r)
    print(u)
    print(runtime)

main()
```

Python code for MMPC control law lgorithm 1

```
import numpy as np
from scipy.optimize import minimize
import time


global N, h, h2, nx, psi_d, init, ny, init_psi, vel
N = 10
h  = 0.1
nx = 7 #Number of values outer loop
ny = 3 #Number of values inner loop
"""Initial states"""
init = [50.0, 10.0]
init_psi = [np.pi/2, 0.0]
vel = 0

"""Outer loop MPC"""
"""Objective function"""
```

```
def MPC_O(x):
    global N, h, nx
    s_obj = 0
    eps_x_obj = 0
    eps_y_obj = 0
    u1_obj = 0
    u2_obj = 0
    for k in range(N):
        s_obj = s_obj + x[2+k*nx]**2
        eps_x_obj = eps_x_obj + 1000*x[3+k*nx]**2
        eps_y_obj = eps_y_obj + 1000*x[4+k*nx]**2
        u1_obj = u1_obj + 0.1*x[5+k*nx]**2
    return s_obj+eps_x_obj+eps_y_obj+u1_obj

"""Constraint functions"""
def dc1(x):
    global h
    px_0 = init[0]
    psi_0 = np.pi/2
    return x[0]-h*x[5]*np.cos(psi_0)-px_0

def dc2(x):
    global h
    py_0 = init[1]
    psi_0 = np.pi/2
    return x[1]-h*x[5]*np.sin(psi_0)-py_0

def dc_px(x):
    global N, h, nx
    px_cons = np.zeros(N-1)
    for k in range(1, N):
        px_cons[k-1] = x[k*nx] - x[(k-1)*nx] - h*x[5+k*nx]*np.cos(x[6+(k
    return px_cons

def dc_py(x):
    global N, h, nx
    py_cons = np.zeros(N-1)
    for k in range(1, N):
        py_cons[k-1] = x[1+k*nx] - x[1+(k-1)*nx] - h*x[5+k*nx]*np.sin(x[
    return py_cons


def pc_px(x):
    global N, nx
    px_cons = np.zeros(N)
```

```
    for k in range(N):
        px_cons[k] = x[k*nx] - 2*x[2+k*nx] - x[3+k*nx]
    return px_cons

def pc_py(x):
    global N, nx
    py_cons = np.zeros(N)
    for k in range(N):
        py_cons[k] = x[1+k*nx] - 1.5*x[2+k*nx] - x[4+k*nx]
    return py_cons

def initial(x, y): #Sets an initial search value for the first iteration
    global N, nx
    init = np.zeros(N*nx)
    for k in range(N):
        init[k*nx] = x
        init[1+k*nx]  = y
        init[2+k*nx] = 15.0
        init[3+k*nx] = 0.0
        init[4+k*nx] = 0.0
        init[5+k*nx] = 5.0
        init[6+k*nx] = np.pi/2
    return init

def bounds(): #Defines inequality constraints
    global N
    bound = []
    for k in range(N):
        bound.append((-100.0, 100.0)) #bound for px at timestep k
        bound.append((-100.0, 100.0)) #bound for py at timestep k
        bound.append((0, 30)) #bound for s at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_x at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_y at timestep k
        bound.append((0.0, 5.0)) #bound for u at timestep k
        bound.append((0, 2*np.pi)) #bound for psi at timestep k
    return bound
bnds = tuple(bounds())


"""Defines constraints in order"""
con1 = {'type': 'eq', 'fun': dc1}
con2 = {'type': 'eq', 'fun': dc2}
con3 = {'type': 'eq', 'fun': dc_px}
con4 = {'type': 'eq', 'fun': dc_py}
con5 = {'type': 'eq', 'fun': pc_px}
```

```
con6 = {'type': 'eq', 'fun': pc_py}
cons = [con1, con2, con3, con4, con5, con6]

"""Inner loop MPC"""
"""Objective function"""
def MPC_I(x):
    global N, ny, psi_d
    psi_obj = 0
    r_obj = 0
    gamma_obj = 0
    for k in range(N):
        psi_obj = psi_obj + (np.pi-x[k*ny])**2
        r_obj = r_obj + 0.01*x[1+k*ny]**2
        gamma_obj = gamma_obj + 0.1*x[2+k*ny]**2
    return psi_obj+r_obj+gamma_obj

"""Constraint functions"""
def dc3(x):
    global h, init_psi
    psi_0 = init_psi[0]
    return x[0]-h*x[1]-psi_0

def dc4(x):
    global h, vel, init_psi
    r_0 = init_psi[1]
    return x[1]-r_0+h*1528/1074*r_0*abs(r_0)-vel*h*37.9*1.26*abs(vel)*np

def dc_psi(x):
    global N, h, ny
    psi_cons = np.zeros(N-1)
    for k in range(1, N):
        psi_cons[k-1] = x[k*ny] - x[(k-1)*ny] - h*x[1+(k-1)*ny]
    return psi_cons

def dc_r(x):
    global N, h, ny, vel
    r_cons = np.zeros(N-1)
    for k in range(1, N):
        r_cons[k-1] = x[1+k*ny] -x[1+(k-1)*ny]+h*1528/1074*x[1+(k-1)*ny]
    return r_cons

def initial_psi(psi, r): #Sets an initial search value for the first ite
    global N, ny
    init = np.zeros(N*ny)
    for k in range(N):
```

```
        init[k*ny] = psi
        init[1+k*ny]  = r
        init[2+k*ny] = 0.0
    return init

def psi_bounds(): #Defines inequality constraints
    global N
    bound = []
    for k in range(N):
        bound.append((0, 2*np.pi)) #bound for psi at timestep k
        bound.append((-np.pi/2, np.pi/2)) #bound for r at timestep k
        bound.append((-35*np.pi/180, 35*np.pi/180)) #bound for steering
    return bound
psi_bnds = tuple(psi_bounds())

"""Defines constraints in order"""
con7 = {'type': 'eq', 'fun': dc_psi}
con8 = {'type': 'eq', 'fun': dc_r}
con9 = {'type': 'eq', 'fun': dc3}
con10 = {'type': 'eq', 'fun': dc4}
psi_cons = [con9, con10, con7, con8]

def simulate(x, y,  psi, r, u, gamma): #Simulate the system (euler metho
    global h2
    D = 1528
    mu_f = 37.9
    R = 1.26
    I_z = 1074
    x = x+h*u*np.cos(psi)
    y = y+h*u*np.sin(psi)
    psi = psi +h2*r
    r = r - h*D*r*abs(r)/I_z + h*R*mu_f*u*abs(u)*np.sin(gamma)/I_z
    return x, y, psi, r

"""SOLVE"""
start=time.clock()
def main():
    global init, init_psi, vel
    tf = 350 #Set simulation time
    """Define empty arrays for the states, inputs and run time"""
    x = []
    y = []
    psi = []
    r = []
    u = []
```

```
gamma = []
runtime = []
runtime2 = []
"""Get initial states"""
x0 = initial(init[0], init[1])
psi0 = initial_psi(init_psi[0], init_psi[1])
x.append(init[0])
y.append(init[1])
psi.append(init_psi[0])
r.append(init_psi[1])
for i in range(1, tf):#Simulate system for the defined simulation ti
    global psi_d
    start=time.clock() #Start runtime timer of outer loop
    sol = minimize(MPC_O,x0,method='SLSQP', bounds=bnds, constraints
    u.append(sol.x[5]) #Get velocity input
    vel = u[i-1]
    psi_d = sol.x[6] #Get desired heading
    runtime.append(time.clock()-start) #End runtime timer of outer l
    start2 = time.clock() #Start runtime timer of outer loop
    sol_in = minimize(MPC_I,psi0,method='SLSQP', bounds=psi_bnds, co
    gamma.append(sol_in.x[2])
    runtime2.append(time.clock()-start2)
    """Define new initial search values"""
    x0[0:(N-1)*nx] = sol.x[nx:N*nx+1]
    x0[(N-1)*nx:N*nx+1] = sol.x[(N-1)*nx:N*nx+1]
    psi0[0:(N-1)*ny] = sol_in.x[ny:N*ny+1]
    psi0[(N-1)*ny:N*ny+1] = sol_in.x[(N-1)*ny:N*ny+1]

    x1, y1, psi1, r1 = simulate(x[i-1], y[i-1], psi[i-1], r[i-1], u[
    """Update new initial values"""
    init = [x1, y1]
    init_psi = [psi1, r1]
    x.append(x1)
    y.append(y1)
    psi.append(psi1)
    r.append(r1)
    print(i)
print(x)
print(y)
print(psi)
print(r)
print(u)
print(runtime)
print(runtime2)
```

```
main()
```

Python code for MMPC control law algorithm 2

```
import numpy as np
from scipy.optimize import minimize
import time


global N, h, h2, nx, psi_d, init, ny, init_psi, vel
N = 10
h = 1
h2  = 0.1
nx = 7 #Number of values outer loop
ny = 3 #Number of values inner loop
"""Initial states"""
init = [50.0, 10.0]
init_psi = [np.pi/2, 0.0]
vel = 0

"""Outer loop MPC"""
"""Objective function"""
def MPC_O(x):
    global N, h, nx
    s_obj = 0
    eps_x_obj = 0
    eps_y_obj = 0
    u1_obj = 0
    u2_obj = 0
    for k in range(N):
        s_obj = s_obj + x[2+k*nx]**2
        eps_x_obj = eps_x_obj + 1000*x[3+k*nx]**2
        eps_y_obj = eps_y_obj + 1000*x[4+k*nx]**2
        u1_obj = u1_obj + 0.1*x[5+k*nx]**2
    return s_obj+eps_x_obj+eps_y_obj+u1_obj

"""Constraint functions"""
def dc1(x):
    global h
    px_0 = init[0]
    psi_0 = np.pi/2
    return x[0]-h*x[5]*np.cos(psi_0)-px_0

def dc2(x):
    global h
    py_0 = init[1]
    psi_0 = np.pi/2
```

```
    return x[1]-h*x[5]*np.sin(psi_0)-py_0

def dc_px(x):
    global N, h, nx
    px_cons = np.zeros(N-1)
    for k in range(1, N):
        px_cons[k-1] = x[k*nx] - x[(k-1)*nx] - h*x[5+k*nx]*np.cos(x[6+(k
    return px_cons

def dc_py(x):
    global N, h, nx
    py_cons = np.zeros(N-1)
    for k in range(1, N):
        py_cons[k-1] = x[1+k*nx] - x[1+(k-1)*nx] - h*x[5+k*nx]*np.sin(x[
    return py_cons


def pc_px(x):
    global N, nx
    px_cons = np.zeros(N)
    for k in range(N):
        px_cons[k] = x[k*nx] - 2*x[2+k*nx] - x[3+k*nx]
    return px_cons

def pc_py(x):
    global N, nx
    py_cons = np.zeros(N)
    for k in range(N):
        py_cons[k] = x[1+k*nx] - 1.5*x[2+k*nx] - x[4+k*nx]
    return py_cons

def initial(x, y): #Sets an initial search value for the first iteration
    global N, nx
    init = np.zeros(N*nx)
    for k in range(N):
        init[k*nx] = x
        init[1+k*nx]  = y
        init[2+k*nx] = 15.0
        init[3+k*nx] = 0.0
        init[4+k*nx] = 0.0
        init[5+k*nx] = 5.0
        init[6+k*nx] = np.pi/2
    return init

def bounds(): #Defines inequality constraints
```

```
    global N
    bound = []
    for k in range(N):
        bound.append((-100.0, 100.0)) #bound for px at timestep k
        bound.append((-100.0, 100.0)) #bound for py at timestep k
        bound.append((0, 30)) #bound for s at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_x at timestep k
        bound.append((-100.0, 100.0)) #bound for eps_y at timestep k
        bound.append((0.0, 5.0)) #bound for u at timestep k
        bound.append((0, 2*np.pi)) #bound for psi at timestep k
    return bound
bnds = tuple(bounds())


"""Defines constraints in order"""
con1 = {'type': 'eq', 'fun': dc1}
con2 = {'type': 'eq', 'fun': dc2}
con3 = {'type': 'eq', 'fun': dc_px}
con4 = {'type': 'eq', 'fun': dc_py}
con5 = {'type': 'eq', 'fun': pc_px}
con6 = {'type': 'eq', 'fun': pc_py}
cons = [con1, con2, con3, con4, con5, con6]

"""Inner loop MPC"""
"""Objective function"""
def MPC_I(x):
    global N, ny, psi_d
    psi_obj = 0
    r_obj = 0
    gamma_obj = 0
    for k in range(N):
        psi_obj = psi_obj + (np.pi-x[k*ny])**2
        r_obj = r_obj + 0.01*x[1+k*ny]**2
        gamma_obj = gamma_obj + 0.1*x[2+k*ny]**2
    return psi_obj+r_obj+gamma_obj

"""Constraint functions"""
def dc3(x):
    global h, init_psi
    psi_0 = init_psi[0]
    return x[0]-h2*x[1]-psi_0

def dc4(x):
    global h2, vel, init_psi
    r_0 = init_psi[1]
```

```
    return x[1]-r_0+h2*1528/1074*r_0*abs(r_0)-vel*h2*37.9*1.26*abs(vel)*

def dc_psi(x):
    global N, h, ny
    psi_cons = np.zeros(N-1)
    for k in range(1, N):
        psi_cons[k-1] = x[k*ny] - x[(k-1)*ny] - h2*x[1+(k-1)*ny]
    return psi_cons

def dc_r(x):
    global N, h, ny, vel
    r_cons = np.zeros(N-1)
    for k in range(1, N):
        r_cons[k-1] = x[1+k*ny] -x[1+(k-1)*ny]+h2*1528/1074*x[1+(k-1)*ny

def initial_psi(psi, r): #Sets an initial search value for the first ite
    global N, ny
    init = np.zeros(N*ny)
    for k in range(N):
        init[k*ny] = psi
        init[1+k*ny]  = r
        init[2+k*ny] = 0.0
    return init

def psi_bounds(): #Defines inequality constraints
    global N
    bound = []
    for k in range(N):
        bound.append((0, 2*np.pi)) #bound for psi at timestep k
        bound.append((-np.pi/2, np.pi/2)) #bound for r at timestep k
        bound.append((-35*np.pi/180, 35*np.pi/180)) #bound for steering
    return bound
psi_bnds = tuple(psi_bounds())

"""Defines constraints in order"""
con7 = {'type': 'eq', 'fun': dc_psi}
con8 = {'type': 'eq', 'fun': dc_r}
con9 = {'type': 'eq', 'fun': dc3}
con10 = {'type': 'eq', 'fun': dc4}
psi_cons = [con9, con10, con7, con8]

def simulate(x, y,  psi, r, u, gamma): #Simulate the system (euler metho
    global h2
    D = 1528
```

```
    mu_f = 37.9
    R = 1.26
    I_z = 1074
    x = x+h2*u*np.cos(psi)
    y = y+h2*u*np.sin(psi)
    psi = psi +h2*r
    r = r - h2*D*r*abs(r)/I_z + h2*R*mu_f*u*abs(u)*np.sin(gamma)/I_z
    return x, y, psi, r

"""SOLVE"""
start=time.clock()
def main():
    global init, init_psi, vel
    tf = 350 #Set simulation time
    """Define empty arrays for the states, inputs and run time"""
    x = []
    y = []
    psi = []
    r = []
    u = []
    gamma = []
    runtime = []
    runtime2 = []
    """Get initial states"""
    x0 = initial(init[0], init[1])
    psi0 = initial_psi(init_psi[0], init_psi[1])
    x.append(init[0])
    y.append(init[1])
    psi.append(init_psi[0])
    r.append(init_psi[1])
    for i in range(1, tf):#Simulate system for the defined simulation ti
        global psi_d
        start=time.clock() #Start runtime timer of outer loop
        sol = minimize(MPC_O,x0,method='SLSQP', bounds=bnds, constraints
        u.append(sol.x[5]) #Get velocity input
        vel = u[i-1]
        psi_d = sol.x[6] #Get desired heading
        runtime.append(time.clock()-start) #End runtime timer of outer l
        x0[0:(N-1)*nx] = sol.x[nx:N*nx+1]
        x0[(N-1)*nx:N*nx+1] = sol.x[(N-1)*nx:N*nx+1]
        for j in range(10):
            start2 = time.clock() #Start runtime timer of outer loop
            sol_in = minimize(MPC_I,psi0,method='SLSQP', bounds=psi_bnds
            gamma.append(sol_in.x[2])
            runtime2.append(time.clock()-start2)
```

```
            """Define new initial search values"""

            psi0[0:(N-1)*ny] = sol_in.x[ny:N*ny+1]
            psi0[(N-1)*ny:N*ny+1] = sol_in.x[(N-1)*ny:N*ny+1]

            x1, y1, psi1, r1 = simulate(x[i-1], y[i-1], psi[i-1], r[i-1
        """Update new initial values"""
            init = [x1, y1]
            init_psi = [psi1, r1]
            x.append(x1)
            y.append(y1)
            psi.append(psi1)
            r.append(r1)
    print(x)
    print(y)
    print(psi)
    print(r)
    print(u)
    print(runtime)
    print(runtime2)

main()
```