Heidi-Irene Harvey Sollie

Linear Model Predictive Control of a 3 DOF Helicopter

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
June 2019

NTNU
Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

Master's thesis

NTNU
Norwegian University of
Science and Technology

Heidi-Irene Harvey Sollie

Linear Model Predictive Control of a 3 DOF Helicopter

Master's thesis in Cybernetics and Robotics
Supervisor: Lars Struen Imsland
June 2019

Norwegian University of Science and Technology
Faculty of Information Technology and Electrical Engineering
Department of Engineering Cybernetics

NTNU
Norwegian University of
Science and Technology

Preface

This thesis addresses the application of model predictive control (MPC) to a bench top model helicopter, of which the university owns ten. The work presented in this thesis has been carried out at the Department of Engineering Cybernetics at the Norwegian University of Science and Technology.

The initial plan for my thesis was a different, more theoretical study, that involved researching and simulating parametrized tube-based MPC. This project would have been based on the work done last semester where tube-based MPC was the focus. However, after a month of researching, my advisor and I came to the conclusion that the project required more time than was originally assumed. Consequently, six weeks after the beginning of the masters, the scope and goal of the thesis shifted drastically.

The new thesis became much more practical, and the first idea was trying both linear model predictive control and tube-based model predictive control, to build on some of the work done last semester. However, tube-based model predictive control with the number of states required to control the helicopter is too computationally expensive when using the implementation presented in last years project report. Therefore, this thesis focuses on the implementation of linear model predictive control of Quanser's 3 DOF helicopter.

The motivation for choosing this project was mostly wanting to apply my work from the previous semester to a practical example. In addition, the practical work in the course TTK4135 Optimization and Control involves the helicopter lab equipment. Up til now, model predictive control of the helicopter has not been a part of this work, despite a large portion of the curriculum involving this topic. Hopefully, the work presented in this

thesis can give students a hands-on experience of model predictive control of dynamic systems.

All work has been carried out at the university, in the rooms where the helicopters are located. The basis for the MATLAB simulation was developed last semester. The work has been conducted using MATLAB 2015b, Simulink and QUARC on the computer in the helicopter lab. The QP solver OSQP was used for the optimization. The basis for the Simulink model is a part of the lab-handout to students in TTK4135.

First and foremost, I would like to thank my advisor, Professor Lars Struen Imsland, for invaluable guidance and for being as excited about the work that was done as I was. I would also like to thank Joakim Rostrup Andersen for occasionally stopping by the lab and helping out, being available for questions and generally showing interest and enthusiasm for the project. In addition, the developers of OSQP who answer Google Forum questions within the hour deserve a mention.

Finally, I would like to thank my family for having supported me throughout these five years. Although not every subject and project has been understood, I have always gotten the support I have needed from you all. And especially, thank you so much to my father for being a motivational source and a great inspiration. I also want to thank Knut both for moral support throughout this process, and for being the only person I know willing to read a master thesis over ten times.

Abstract

This thesis presents the implementation of linear model predictive control of a model helicopter with three degrees of freedom (DOF). It is a bench-top model of a propeller-actuated helicopter where the pitch (roll) and elevation (vertical) of the helicopter are used as control inputs to steer the system to its equilibrium from an arbitrary starting point in travel (yaw). A pitch controller (PD) and elevation controller (PID) make up the lower level of control.

A nonlinear model of the helicopter dynamics is developed and linearized to state-space representation. The entire control architecture, along with the C library for the QP solver, is implemented in Simulink. The real-time control software QUARC combined with the code generation capabilities of Simulink, allows for this model being run on the I/O board connected to the helicopter.

Linear MPC with and without a terminal cost and terminal constraint is presented, first in a numerical simulation of the linearized model and then on the physical helicopter. Furthermore, a known constant drift of the helicopter is modeled and a Kalman filter is implemented to remove the constant offset in travel that this drift causes.

First and foremost, this thesis is proof-of-concept for the model predictive control of a dynamic 3 DOF helicopter. It is demonstrated that using MPC for such a dynamic system is possible with modern QP solvers and advanced embedded hardware. The QP solver performs the online computation with an average solve time of about 30 milliseconds, which is frequent enough for stable control.

Sammendrag

Denne oppgaven presenterer implementasjonen av lineær modellprediktiv regulering (MPR) av et modellhelikopter med tre frihetsgrader, der pitch og høyden til helikopteret er brukt som pådrag for å styre systemet til likevektspunktet fra et vilkårlig startpunkt. En pitch-regulator (PD) og høyde-regulator (PID) utgjør det laveste nivået av reguleringsarkitekturen.

En ulineær modell av helikopterdynamikken er utviklet og linearisert rundt likevektspunktet, og uttrykt i tilstandsromrepresentasjon. Hele reguleringsarkitekturen, i tillegg til C-biblioteket til QP-løseren, er implementert i Simulink. Sanntidsprogramvaren QUARC kombinert med kodegenerasjonsegenskapene til Simulink gjør det mulig for denne modellen å kjøre direkte på I/O-kortet som styrer helikopteret.

Lineær MPR med og uten terminalkostnad og terminalbeskrankning presenteres først i en numerisk simulering av helikoptermodellen og deretter på det fysiske helikopteret. Videre er en konstant forstyrrelse på helikopteret modellert, og et Kalmanfilter er implementert for å fjerne avviket denne driften forårsaker.

Først og fremst er denne oppgaven en testimplementasion av modellprediktiv regulering på et dynamisk helikopter med tre frihetsgrader. Det er demonstrert at bruk av MPR for et slikt dynamisk system er mulig med nye QP-løsere og moderne teknologi. QP-løseren utfører optimalseringsberegninger i sanntid med en gjennomsnittlig løsetid på rundt 30 millisekunder, noe som er hyppig nok til å sørge for stabil kontroll.

Table of Contents

Preface **i**

Abstract **iii**

Sammendrag **v**

1 Introduction **3**
1.1 Background 3
1.1.1 Motivation 5
1.1.2 Outline 5

2 Literature Review **7**
2.1 Quanser's 3 DOF Helicopter 7
2.2 Embedded Optimization 9

3 Model Predictive Control Theory **11**
3.1 Open-loop optimal control problem 12
3.2 Terminal cost and terminal constraint 15
3.3 Integral action in MPC 18
3.3.1 Kalman filter 19
3.4 Slack variables 20
3.5 Stability of MPC 20
3.6 Robustness of MPC 21

4 Quadratic Programming Solver **23**

5 Developing a process model **27**
5.1 Helicopter model 27
5.2 Selection of constraints 34
5.3 Modeling the constant disturbance 34

6 Hardware and Software Implementation **37**
6.1 OSQP 37
6.2 Hardware-in-the-loop (HIL) 38
6.3 Implementing the optimization problem for OSQP 40
6.4 Simulink Model 43
6.4.1 MPC 44
6.4.2 Estimator 49
6.5 Multi-Parametric Toolbox 49

7 Results **51**
7.1 Numerical simulation of helicopter model 52
7.2 Experimental results of helicopter performance 60

8 Discussion **73**
8.1 Future work 77

9 Conclusion **79**

Acronyms **81**

Bibliography **81**

Appendices

A System parameters

B MATLAB Code

C Simulink model

D Additional figures from results

List of Figures

2.1 Quanser's 3 DOF helicopter 7

3.1 Visualization of MPC, from Foss and Heirung [9] 12

4.1 Computation time versus problem dimension for OSQP and GUROBI for 7 benchmark problem classes [27] 25

5.1 Diagram of helicopter forces, angles and physical sizes 28
5.2 Diagram of system control architecture 32

6.1 Information flow in Hardware-in-the-loop (HIL) testing 38
6.2 A sliced plot of the terminal set $\mathbb{X}_f$ 43
6.3 Complete Simulink model 44
6.4 Simulink model of MPC block 44
6.5 Simulink model of estimator 49

7.1 Simulation of nominal MPC 53
7.2 Simulation of stable MPC 54
7.3 Simulation of nominal MPC with longer horizon 55
7.4 Simulation of stable MPC with Kalman filter 57
7.5 Estimation of disturbance from simulation 58
7.6 Online performance of nominal MPC 61
7.7 OSQP solve time of nominal MPC 62
7.8 Online performance of stable MPC 63
7.9 OSQP solve time of stable MPC 64
7.10 Online performance of stable MPC with Kalman filter 65
7.11 OSQP solve time of stable MPC with Kalman filter 66
7.12 Online performance of stable MPC with frequency 25 Hz . . . 67
7.13 OSQP solve time of stable MPC with frequency 25 Hz 68
7.14 Online performance of stable MPC with frequency 5 Hz . . . 69

7.15 Estimated disturbance from Kalman filter 70

C.1 Simulink model . . . . . .
C.2 Helicopter interface . . . . . .
C.3 Pitch controller . . . . . .
C.4 Elevation controller . . . . . .
C.5 Voltage conversion block . . . . . .
C.6 R6 -> R7 block . . . . . .

D.1 Simulation of nominal MPC with Kalman filter . . . . . .
D.2 OSQP solve time of simulation of nominalMPC with Kalman filter . . . . . .
D.3 Online performance of nominal MPC with Kalman filter . . .
D.4 OSQP solve time of nominal MPC and Kalman filter . . . . .
D.5 OSQP solve time of stable MPC with frequency 5 Hz . . . . .

Listings

6.1 C code in the S-Function Builder block 45
6.2 timer.h 47
6.3 Exerpt from gen_code_params.m 48
B.1 execute.m . . . . . .
B.2 offline_calc.m . . . . . .
B.3 kalman_filter.m . . . . . .
B.4 gen_mpc_matrices.m . . . . . .
B.5 gen_code_osqp.m . . . . . .

Nomenclature

$A \in \mathbb{R}^{n \times m}$	A is a matrix with $(n \times m)$-dimensions with real numbers
A^{col_n}	The n-th column of matrix A
A^{row_n}	The n-th row of matrix A
$\mathbb{Z}_{0:N}$	The set of all integers from 0 to N
p	pitch
e	elevation
λ	travel
ϵ_p	slack variable
ϕ	$\begin{bmatrix} \mathbf{u} \\ \epsilon_{\mathbf{p}} \end{bmatrix}$
$F(\cdot)$	terminal cost function
N	optimization horizon
$V(\cdot)$	finite horizon cost function
$\mathbf{u}$	$\begin{bmatrix} u_0 & u_1 & \dots & u_{N-1} \end{bmatrix}^\top$
$\mathbf{x}$	$\begin{bmatrix} x_1 & x_2 & \dots & x_N \end{bmatrix}^\top$
$\mathbb{P}_N(\cdot)$	online optimization problem
$\mathbb{U}$	input constraint set
$\mathbb{X}$	state constraint set
$\mathbb{X}_f$	terminal set

Chapter 1

Introduction

1.1 Background

In recent years, propeller-actuated systems have gained popularity, mainly due to their increased availability and the advancement of cheaper and more lightweight electronics. Due to the under-actuated[1] nature of a lot of these types of systems, they make for a challenging control problem. Quanser's 3 degrees of freedom (DOF) helicopter is a bench-top model of a dual-propeller actuated helicopter and is an under-actuated multi-input multi-output (MIMO) system with nonlinear characteristics. With the helicopter and the Hardware-in-the-loop (HIL) data board, as well as the Simulink software interface, implementing control techniques on the helicopter is possible.

The helicopter is made up of three rotational joints, with the end bar carrying a pair of propellers actuated by DC motors. The joint angles are supplied by incremental encoders, and the propellers are driven by two power amplifiers. The helicopter is mounted in a slip ring which allows for 360-degree movement in the travel angle. There are numerous control implementations presented for this type of system (a 3 DOF helicopter) from the past two decades.

However, although a few optimization based control implementations have been presented, a minority have actually been tested on the physical system. Additionally, there have been no publications of an actual implementation

[1] fewer actuators that degrees of freedom

of MPC involving online computations on the Quanser helicopter. The work presented in this thesis demonstrates that due to the advancement of processing power for embedded hardware and the increased performance of available QP solvers, model predictive control of a 3 DOF helicopter is in fact possible.

Model predictive control (MPC) is an optimization based repetitive control algorithm that solves a finite horizon open-loop optimal control problem online and applies the first control input to the system at every time step. It was first advocated by Richalet et al. [24] in the late 1970s[2] and quickly became widespread mainly in the process and petrochemical industry. The reason for this was its ability to handle control of systems with hard constraints, where offline computation of a control law is difficult, and it is also easily applied to MIMO systems. MPC is said to be the only advanced control technology to have a significant impact on industrial process control [15].

The idea of optimizing the control input by solving an optimal control problem was not unique at the time. However, MPC differed from other control methods by solving the control problem online for the current state of the system, rather than using the control problem offline to develop an explicit feedback law. The computational complexity involved with the online calculation of MPC has been the bottleneck when determining which systems to implement it on. With advancing technology and the increasing demand for smaller and faster electronic devices, the application of MPC has been able to move into areas such as robotics, mechatronics, and aerial vehicles. This involves the concept of embedded optimization, or more specifically, embedded MPC, where the optimal control problem must be solved online on embedded hardware. Since the migration of MPC into the field of dynamic system is relatively new, attitude regulation of helicopters using MPC is a field not thoroughly explored.

The implementation of model predictive control of the helicopter is made possible by OSQP, an open-source general-purpose QP solver presented in 2018 by Stellato et al. [27]. The benchmark results for this algorithm shows that it outperforms most open-source and commercial QP solvers available, in terms of computational time. The code generation software package also available with OSQP generates tailored C code for a specific problem that can

[2]though earlier proposals do exist, such as Propoi (1963) [17]

run on embedded platforms [2]. A Simulink model of the HIL architecture containing the C code from OSQP, along with QUARC, a real-time control software developed by Quanser, enables this model to run on the embedded hardware driving the physical system.

1.1.1 Motivation

The rapid advancement in the field of embedded optimization means that the use of computationally expensive controllers such as MPC can be used to control highly nonlinear systems with fast dynamics such as a 3 DOF helicopter. And due to the increasing availability of small scale dynamic systems for didactic purposes, the development of such implementations is easier than ever.

The motivation for this thesis is a proof-of-concept demonstration of linear model predictive control of a helicopter. This will allow the further development of model predictive control of similar systems.

1.1.2 Outline

First, in Chapter 2, a literature review regarding control methods based on or implemented on the Quanser Helicopter will be presented, along with an analysis of the embedded optimization of dynamic systems with model predictive control. This is followed by model predictive control theory as well as a robustness and stability analysis, in Chapter 3. Thereafter, in Chapter 4, the algorithm used in the QP solver, OSQP, and an analysis of why it outperforms similar solvers will be presented. Then, both the software and hardware implementation of the controller is described, in Chapter 6. In Chapter 7, a presentation of the results from both numerical simulation and the actual helicopter performance takes place. The performance of the QP solver will also be presented here. Then a discussion of the controller performance and the work described, as well as potential improvements and future work is presented in Chapter 8, before, finally concluding the work in Chapter 9.

Chapter 2

Literature Review

2.1 Quanser's 3 DOF Helicopter

Quanser is a Canadian-based company that develops and supplies a wide range of engineering lab equipment within mechatronics, robotics, and control, designed for an educational setting. Among these is the 3 DOF helicopter, a bench-top model of a tandem rotor helicopter, with 3 degrees of freedom (DOF); pitch (roll), travel (yaw) and elevation (vertical). Figure 2.1, taken from Quanser [19], shows a picture of the equipment.

Figure 2.1: Quanser's 3 DOF helicopter

Since this hardware is designed for testing and developing control laws for systems with similar dynamics, there is a lot of research performed and

available with regard to this system, mainly control implementations based on both a linear and nonlinear model.

Veeraboina and Ordonez [29] presents the design and implementation of numerous different controllers on an Arduino Mega, along with a performance comparison of them. Controllers such as LQR, LQR based PID, IO feedback linearization and a direct adaptive fuzzy controller. Arican et al. [1] presents a state-dependent Riccati equation based optimal control for the nonlinear system, while Kocagil et al. [14] presents a review of state-dependent Riccati equation based optimal control, along with model reference adaptive control and sliding mode control. Just recently, robust adaptive control of multiple helicopters with actuator faults was presented by Yang et al. [30].

None of these implemented control methods mentioned are particularly well equipped for constraint handling, which is an important advantage of MPC. However, there have been some such methods proposed.

For instance, there are a few explicit model predictive control (EMPC) methods presented. EMPC is a version of MPC that reduces the online computational cost of the controller by moving all those calculations offline. This entails providing an explicit optimal control law by calculating all possible optimal input trajectories in the operating region of the system. Although these calculations are more complex than the online calculations of traditional MPC, because the optimal control problem has to be solved for every possible state and input combination, there is no time limit for the offline calculation and it will therefore not affect the performance of the controller online.

EMPC of the helicopter has been presented and tested by numerical simulation and semi-physical simulation by Ju and Xinyan [12]; both feasibility and performance is demonstrated. Cheng et al. [33] presents another EMPC for attitude control and tracking, along with its implementation, and compares the performance with a PID controller.

These authors have successfully implemented a form of MPC on the helicopter by reducing the online computational cost. Despite being an acceptable solution, it requires in general a large memory footprint of the hardware. Additionally, a larger operating area requires a larger memory, to store the explicit control law. For embedded hardware with limited memory

resources, the best solution would be to efficiently solve the optimal control problem online.

Nonlinear MPC (NMPC) for attitude regulation using successive linearization (continous linearization of the nonlinear model based on the current state and control input) has been suggested by Zhai [32], which solves the optimal control problem online for the next control input. Although this controller is shown to outperform linear MPC, it was not actually implemented on the physical system, only tested with numerical simulation. Similarly, the same author suggests successive linearization on a nonlinear model along with an observer to obtain robust control, but this is also only tested in simulation [31].

Up til now, there have been no implementation proposals for model predictive control of this helicopter, and this is therefore a useful and interesting area of research. Since the framework for this project is the work done last semester, Sollie [26], based on linear model predictive control, this is the method that will be used in the implementation presented in this thesis.

2.2 Embedded Optimization

Embedded optimization is the concept of continuously solving an optimization problem on an embedded system, where the data is updated real-time from a system that evolves over time. Naturally, the exact evolution of the system can only be predicted to a certain degree, therefore the optimization should be dynamic, a function of both time and the process model. Embedded optimization is complex, mainly because of the close relationship between the solver, the hardware and the process [13].

In the late 1970s, model predictive control was introduced as a combination of feedback control theory and numerical optimization [23], and quickly gained traction as a constraint handling control method, that could easily be applied to MIMO systems. This algorithm was run on computers at low frequencies, updating the control input as seldom as once a minute [18]. As the computer was popularized a few decades later, optimization methods on embedded hardware were introduced to address needs such as cost-effectiveness and efficient energy use. Advancements have been made for optimization calculations in real-time, and successful implementations onto

for example central processing units, digital signal processors, programmable logic controllers, and field-programmable gate arrays have been presented [8]. Now, this control method originally used in a specific industry could be applied to areas such as robotics, aerospace and mechatronics, and the concept of embedded MPC arose. With this development, challenges related to algorithms and their implementations, as well as the computing hardware arises.

Embedded MPC is an increasingly relevant method, as modern processing power has finally caught up to meet the computational demands that the online calculations require. An implementation of embedded MPC requires low code complexity and a small memory footprint to ensure it runs adequately on the applicable hardware and matches real-time demand. The existing control solutions for embedded systems need a lot of tuning to cope with constraints and nonlinearities. As mentioned, multivariable control for MPC, and the incorporation of state and input constraints is the reason for its rapid spread and a good reason for an implementation on a system such as a 3 DOF helicopter.

Chapter 3

Model Predictive Control Theory

Model predictive control (MPC), also called receding horizon control, is a control strategy that implements an implicit control law by repeatedly solving a constrained optimal control problem over a finite horizon.

The basic idea behind model predictive control is optimizing the control input and resulting state trajectory, given a finite horizon cost function, for every time step and applying the first optimal control input to the system.

MPC is one of the most attractive feedback strategies utilized, especially for constrained linear systems.

The three main stages of the algorithm are

1. Measuring the current state of the system
2. Solving the given optimization problem
3. Applying the first control input to the system, while the others are rejected

Figure 3.1 shows an abstract visualization of the principle of MPC, taken from Foss and Heirung [9]. The optimal state and control trajectories are visualized, and the first control step is applied to the plant.

If a nonlinear process model is used in the prediction, it is referred to as nonlinear model predictive control (NMPC), while a linear model will just be

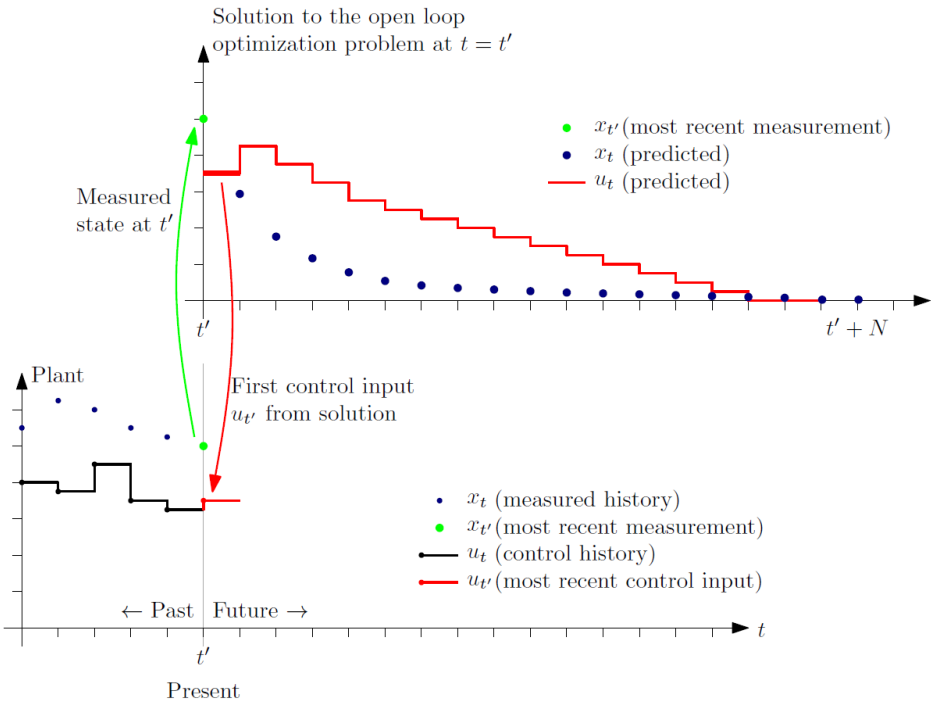

Figure 3.1: Visualization of MPC, from Foss and Heirung [9]

referred to as MPC, throughout this paper. Using nonlinear model predictive control results in a more accurate process model leading to a more accurate prediction, but at the cost of computational cost and run-time.

Linear MPC is by far the most popular of the two. It is less complex [1], and the feedback mechanism of MPC can account for differences between the model and the actual system, which will exist since a linear model will never be completely accurate in an actual process.

3.1 Open-loop optimal control problem

The basic formulation for the open-loop optimal control problem solved online (in which the initial state is the current state of the system) is,

[1] in best case it is a convex QP problem

$$\mathbb{P}_N(x_0) \;=\; \min_{\mathbf{x},\mathbf{u}} \; V(\mathbf{x},\mathbf{u}) \;, \tag{3.1}$$

$$\text{subject to} \quad x_{i+1} = Ax_i + Bu_i \qquad \forall\, i \in \mathbb{Z}_{[0:N-1]} \;, \tag{3.2}$$

$$x_i \in \mathbb{X} \qquad \forall\, i \in \mathbb{Z}_{[1:N]} \;, \tag{3.3}$$

$$u_i \in \mathbb{U} \qquad \forall\, i \in \mathbb{Z}_{[0:N-1]} \;, \tag{3.4}$$

with

$$\mathbf{x} = \begin{bmatrix} x_1 \\ \vdots \\ x_N \end{bmatrix} \;,\; \mathbf{u} = \begin{bmatrix} u_0 \\ \vdots \\ u_{N-1} \end{bmatrix} \;, \tag{3.5}$$

$$\tag{3.6}$$

and

$$x_i, \mathbb{X} \in \mathbb{R}^n \;, \quad u_i, \mathbb{U} \in \mathbb{R}^m \;, \tag{3.7}$$

$$Q \in \mathbb{R}^{n\times n} \;, \quad R \in \mathbb{R}^{m\times m} \;, \tag{3.8}$$

$$A \in \mathbb{R}^{n\times n} \;, \quad B \in \mathbb{R}^{n\times m} \;. \tag{3.9}$$

The goal is to optimize performance by minimizing the finite horizon cost,

$$V(\mathbf{x},\mathbf{u}) = \frac{1}{2}\sum_{k=0}^{N-1} (x_{k+1}^\top Q x_{k+1} + u_k^\top R u_k) \;. \tag{3.10}$$

The matrices A and B are the matrices for the state space model for the given system, and Q and R are the weighting matrices for the state and control input, respectively. The matrices Q and R must be real and symmetric, Q must be positive semidefinite, and R must be positive definite. The state must be detectable through Q[2]. Additionally, the state and input constraint sets $\mathbb{X}$ and $\mathbb{U}$ must be polyhedrons, which ensures the constraints are linear. When these requirements are fulfilled, the optimal control problem is a convex QP problem.

One thing to consider is that because the initial value x_0 is defined and there is a hard constraint $x^+ = Ax + Bu$, the real optimization variable for this

[2] $(A, \sqrt{Q})$, must be detectable

problem is the control input trajectory $\mathbf{u}$ alone, because the resulting state trajectory merely follows the value of x_0 and the state space equation.

Therefore, a common method for reducing the complexity of the online optimization problem is removing the system equation $x^+ = Ax + Bu$ from the equality constraints and only optimizing over the control inputs $\mathbf{u}$.

Observe the following development

$$\begin{aligned} x_1 &= Ax_0 + Bu_0 \ , \\ x_2 &= Ax_1 + Bu_1 \\ &= A^2x_0 + ABu_0 + Bu_1 \ , \\ x_3 &= Ax_2 + Bu_2 \\ &= A^3x_0 + A^2Bu_0 + ABu_1 + Bu_2 \ , \\ &\vdots \\ x_k &= A^{k-1}Bu_0 + A^{k-2}Bu_1 + \ldots + Bu_2 + A^kx_0 \ . \end{aligned} \tag{3.11}$$

Writing this for the whole state trajectory yields,

$$\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix} = \begin{bmatrix} B & 0 & \ldots & 0 & 0 \\ AB & B & \ldots & 0 & 0 \\ & & \vdots & & \\ A^{N-1}B & A^{N-2}B & \ldots & AB & B \end{bmatrix} \begin{bmatrix} u_0 \\ u_1 \\ \vdots \\ u_{N-1} \end{bmatrix} + \begin{bmatrix} A \\ A^2 \\ \vdots \\ A^N \end{bmatrix} x_0 \ , \tag{3.12}$$

or

$$\mathbf{x} = S_u\mathbf{u} + S_x x_0 \ . \tag{3.13}$$

This equation (3.13) will be used to remove all states from the optimization problem, minimizing the complexity of the problem.

In the formulation of the online optimal control problem presented here, the prediction horizon is identical to the control horizon and will be referred to as the optimization horizon N. It is possible to have a lower control horizon than prediction horizon, where the actuator assumes the terminal control value to be constant past the control horizon. This is also a method for reducing online computational complexity.

3.2 Terminal cost and terminal constraint

The optimization problem presented in Section 3.1 has the same characteristics as some of the earliest proposals of MPC. In these versions, stability was not guaranteed, but was achieved through tuning the cost and optimization horizon, and limiting the analysis and applications to stable systems. Later research devoted considerable attention to proposals of modifying the optimal control problem to ensure nominal stability, and one such suggestion was adding a terminal cost $F(x_N)$ and a terminal constraints $x_N \in \mathbb{X}_f$ to the last state of the optimal trajectory [28]. Most recent model predictive controllers belong to this category [22].

Terminal cost.

A disadvantage to the finite horizon optimization problem is precisely that it optimizes over a finite horizon; it does not consider the system states beyond this horizon N. A strategy to achieve better stability properties for the finite horizon problem is to optimize over the infinite horizon cost function, by finding an explicit expression for the cost of the states beyond the horizon.

First, consider the infinite horizon objective function as a sum split into two terms,

$$\sum_{k=0}^{N-1} (x_k^\top Q x_k + u_k^\top R u_k) + \sum_{k=N}^{\infty} (x_k^\top Q x_k + u_k^\top R u_k) \ . \tag{3.14}$$

Let the last term be $F(\cdot)$, the cost beyond the horizon, and the goal is to obtain explicit expression for this term.

The local controller $u = \kappa(x)$ is introduced for the states beyond the horizon, where $\kappa(x) = Kx$, with K being the feedback gain from the LQR, such that

$$\kappa(x_k) = K x_k, \ \ \forall \, k \in \mathbb{Z}_{[N:\infty]} \ , \tag{3.15}$$

and

$$F(\cdot) = \sum_{k=N}^{\infty} x_k^\top (Q + K^\top R K) x_k \ . \tag{3.16}$$

The state feedback in an LQR is

$$K = -(R + B^\top PB)^{-1} B^\top PA \ , \tag{3.17}$$

where P is the solution[3] to the discrete-time algebraic Riccati equation (DARE),

$$A^\top PA - P + Q - A^\top PB(R + B^\top PB)^{-1} B^\top PA = 0 \ . \tag{3.18}$$

Inserting K into the DARE gives,

$$A_K^\top PA_K - P + Q + K^\top RK = 0 \ , \tag{3.19}$$

with $A_K = A + BK$ which also happens to be the augmented system matrix for the closed-loop system

$$x_{k+1} = Ax_k + B(Kx_k), \ k \in \mathbb{Z}_{[N:\infty]} \ . \tag{3.20}$$

Inserting the revised equation (3.19) into (3.16), yields

$$\begin{aligned} F(x_N) &= \sum_{k=N}^{\infty} x_k^\top Px_k - \sum_{k=0}^{\infty} x_{k+1}^\top Px_{k+1} \\ &= x_N^\top Px_N \ . \end{aligned} \tag{3.21}$$

This function is referred to as a *terminal cost* or sometimes *terminal penalty*, and when added to the finite horizon objective function adds a cost to the states beyond the horizon. The literature shows this term by itself can achieve stability, however in these analyses, there is an implicit requirement of a terminal constraint $x_N \in \mathbb{X}_f$ that is satisfied for every initial state x_0 in a given compact set [22].

Terminal constraint.

In Rawlings and Mayne [21], the terminal cost in itself is not considered to materially affect the online problem, but the addition of a terminal constraint can have a significant effect.

Although there are many methods of calculating or approximating this set, $\mathbb{X}_f$, the main idea is to ensure the state trajectory ends in a set with the

[3]For the solution to exist, (A, B) must be stabilizable

property that for any feasible control input, the successor state will also be in the set. This set is known as a *control invariant* set, and will ensure nominal stability past the optimization horizon.

Instead of considering all feasible control inputs, consider all feasible control inputs that satisfy the feedback control law $u = \kappa(x)$, where as previously $\kappa(x) = Kx$. From this, an autonomous system model can be obtained,

$$\begin{aligned} x^+ &= A_K x \ , \\ (x, Kx) &\in (\mathbb{X} \times \mathbb{U}) \ . \end{aligned} \tag{3.22}$$

For autonomous systems, where there is no control input, the set of all feasible states where all succeeding states are feasible and remain in the set is known as a *positively invariant* set.

Definition: A set Ω is positively invariant if and only if for all states in it, the states satisfy the constraints and successor states are still in it. The mathematical definition is given by

$$\forall\, x \in \Omega \ : \ (x, Kx) \in (\mathbb{X} \times \mathbb{U}) \ \wedge \ A_K x \in \Omega \ . \tag{3.23}$$

Furthermore, the best option is to use the *maximal* positive invariant set (MPIS), which is the positively invariant set that contains all other positively invariant sets. To calculate the MPIS, start with the set

$$\Omega_1 = \{(x, Kx) \in (\mathbb{X} \times \mathbb{U}) \ : \ A_K \cdot (x, Kx) \in (\mathbb{X} \times \mathbb{U})\} \ . \tag{3.24}$$

Then define the general set

$$\Omega_i = \{(x, Kx) \in \Omega_{i-1} \ : \ A_K \cdot (x, Kx) \in (\mathbb{X} \times \mathbb{U})\} \ , \tag{3.25}$$

and recompute the sets Ω_i, $i = 1, 2, \ldots$, until the sets converge, i.e. $\Omega_i = \Omega_{i-1}$. When A_K is asymptotically stable[4], this set exists and is non-empty.

The necessity of the terminal constraint is addressed in [21] and is in a sense 'replacement' for a longer optimization horizon, because it gives the same

[4] all eigenvalues are inside the unit circle

effect, along with a lower computational cost. The purpose is to steer the state to $\mathbb{X}_f$ in a finite time, and inside $\mathbb{X}_f$, beyond the horizon, a local stabilizing controller is employed.

3.3 Integral action in MPC

Consider the case where the helicopter has an unmodelled constant disturbance that affects the helicopter dynamics, such that there is a constant deviation present in one or more states. Since the MPC controller introduced in this section does not have integral action, this has to be embedded into the controller, to mitigate the constant offset.

Let the process model with the disturbance be

$$\begin{aligned} x^+ &= Ax + Bu + A_d d \ , \\ y &= Cx + Du + C_d d \ , \end{aligned} \tag{3.26}$$

where A_d and C_d are matrices that model how the disturbance is coupled with the system dynamics and the output.

This disturbance can be incorporated into the system model by including the additional state d, as follows

$$\begin{aligned} \begin{bmatrix} x^+ \\ d^+ \end{bmatrix} &= \begin{bmatrix} A & A_d \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ \mathbf{0} \end{bmatrix} u \ , \\ y &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + Du \ . \end{aligned} \tag{3.27}$$

The system model has now been augmented with the state d with the evolution

$$d^+ = d \ , \tag{3.28}$$

to reflect its constant nature.

An observer can now be implemented to estimate this state d, and when this augmented model is used in the MPC prediction, the optimal control inputs will be selected so this constant offset is minimized.

Rawlings and Mayne [21] discuss the concept of constructing an MPC controller to achieve zero offset and state that this method is "similar

to what one achieves when using the integral mode in PID-control of an unconstrained system" [21].

To estimate the disturbance, a Kalman filter will be implemented.

3.3.1 Kalman filter

For implementing an estimator, the Kalman filter is chosen, otherwise referred to as a linear-quadratic estimator (LQE). The Kalman filter implementation used is supplied by MATLAB and assumes the system model

$$\begin{aligned} x_k &= Ax_k + Bu_k + Gw_k \ , \\ y_k &= Cx_k + Hw_k + v_k \ , \end{aligned} \tag{3.29}$$

where w_k and v_k are the process noise and measurement noise, respectively.

These are assumed to be white Gaussian noise, which have the following statistical distribution,

$$\begin{aligned} E(w_k w_k^\top) &= Q_K \ , \\ E(v_k v_k^\top) &= R_K \ , \end{aligned} \tag{3.30}$$

where Q_K and R_K are covariance matrices for the given disturbances. These can either be measured or chosen and tuned, depending on the estimator performance. With this, the Kalman filter calculates an observer gain matrix K_f that minimizes the variance of the estimation error.

As opposed to other observers like the Luenberger observer (which was initially implemented but did not perform well) the tuning parameters for the Kalman filter have a much more intuitive meaning. For the Luenberger observer the estimator gain is found by selecting poles of the closed-loop matrix of the error dynamics $(A - LC)$, which can be a difficult process. But when tuning the Kalman filter, the tuning parameters are the values in the process noise covariance matrix Q_K and the measurement noise covariance matrix R_K. So for instance, a change in the measurement noise of a certain output would change the smoothness of this estimation, because a higher measurement noise would mean the filter depends less on this output measurement.

3.4 Slack variables

A common issue when implementing MPC, is the possibility that the system enters into a state which yields the optimization problem infeasible. This is common when there is a disturbance in the system or an inaccurate model is used, such that the system dynamics varies from the prediction of the MPC controller.

In this case, a slack variable ϵ can be added to constraints, to transform the constraints from *hard* to *soft* constraints. This slack variable is then added to the cost function to incentivize the MPC to break the hard constraints only if necessary.

Regarding slack variables in MPC, one can also reduce complexity by using the same variable for the whole horizon. However, this leads to the case where breaking the constraint at one time is just as costly as breaking the constraint for the whole horizon.

3.5 Stability of MPC

The earliest versions of MPC, for example presented in Richalet et al. [25], did not guarantee stability. This was achieved through careful tuning of the cost function and restricting attention to stable plants, and choosing a large enough optimization horizon. Thus, one could achieve similar stability properties as an infinite horizon, where guaranteed stability has been demonstrated.

As was mentioned above, a modification of the optimal control problem that guaranteed stability is the addition of the terminal cost and terminal constraint.

The following conditions on the terminal cost and constraint are presented and proved to ensure closed-loop asymptotic (exponential) stability [22],

1. $\mathbb{X}_f \subset \mathbb{X}$, is closed and contains the origin ,
2. $\kappa(x) \in \mathbb{U} \, \forall \, x \in \mathbb{X}_f$,
3. $\mathbb{X}_f$ is a positively invariant set under $\kappa(x)$,
4. $F(x_N)$ is a local Lyapunov function .

When the specific control problem for this system is presented, it will be shown that these conditions are met.

3.6 Robustness of MPC

Robustness, in the context of control theory, has to do with the controllers stability with regard to uncertainty or model inaccuracies. Robust control methods function properly when exposed to uncertainty, provided the uncertainty are found within some set. The earliest analyses of the robustness of MPC can be found in Richalet et al. [25]. They used impulse response models to investigate robustness in the case of gain mismatch.

There are several approaches to studying robustness of a system. One can attempt to characterize the disturbance, and then consider all possible realizations of this disturbance. Or one can consider the robustness of a closed-loop system, designed using the *nominal system*, the system without disturbance. This is called *inherent* robustness and the analysis presented here will focus on this. The robustness of receding-horizon control has been investigated by de Nicolao et al. [6].

It is shown that the robustness of the system,

$$x^+ = f_a(x) = Ax + Bu + Gw \ , \tag{3.31}$$

is guaranteed for any disturbance w lying in the set,

$$D_a = \{w : V(f_a(x^*)) < V(x^*) \, \forall \, x^* \neq 0 \in \mathbb{R}^n\} \ , \tag{3.32}$$

where $V(x^*)$ is the optimal value of the cost function for the problem at any given time.

In the case of the Quanser helicopter, there are many unmodelled uncertainties, the main one being the nonlinear characteristics of the system. Another uncertainty discussed further in section 5.3, is a constant force affecting the travel of the helicopter.

Since there is no way of guaranteeing that all uncertainties adhere to the condition of being contained in the set D_a, the control presented in this thesis is not completely robust. However, the conclusion will be that the closed-loop control of the helicopter is *somewhat* robust, since the uncertainty will at times be contained in the given set.

One approach to robust model predictive control is tube-based MPC. This is a variation that ensures that the state trajectory will always be contained in a tube centered around the nominal state trajectory, given that the assumptions on the disturbance hold, by tightening the state and control constraints.

The resulting control problem when attempting to control the, given system using this model and the implementation developed in Sollie [26], is too complex, with regard to both problem dimension and computational cost, so this idea was abandoned quite early in the process.

Chapter 4

Quadratic Programming Solver

In order to implement MPC, an open-loop optimal control problem must be solved online for the controller to obtain its next control input. This problem must be solved quite efficiently, and to a high degree of accuracy, which means that the selection and implementation of an optimization algorithm is important. The control problem is a quadratic-programming problem (QP problem), which is a linearly constrained quadratic optimization problem.

A new QP solver was presented in 2018 by Stellato et al. [27], and has gained traction since. The main algorithm is based on the alternating direction method of multipliers (ADMM), and is a general purpose solver for quadratic programs. The algorithm carries out an initial matrix factorization, and is beyond that division-free, which makes it suitable for real-time applications, such as embedded optimization.

ADMM solves convex optimization problems using a divide-and-conquer method by breaking the problem up into smaller pieces. It takes the iterative method from the augmented Lagrange method and forces decomposability, giving it a smoother convergence but also a faster run-time.

The implementation solves the following QP problem,

$$\begin{aligned} \min \quad & \frac{1}{2} z^\top P z + q^\top z \\ \text{subject to} \quad & l \leq Az \leq u \ , \end{aligned} \tag{4.1}$$

with a positive semidefinite matrix P and optimization variable z.

The ADMM algorithm used in this implementation is presented by Boyd et

al. [5]. The initial QP problem is rewritten by adding auxiliary variables to an equality constrained QP problem and this formulation is then used in the ADMM algorithm. There are step-size parameters $\sigma, \rho > 0$ to determine step length and the relaxation parameter $\alpha \in (0, 2)$.

When solving the equality constrained reformulation of the problem, both a direct and indirect method can be used. If the direct method is employed, a factorization of the KKT matrix is performed prior to the first iteration, because the matrix is identical for every iteration. When the factorization cost is much higher than the solve cost, this method is useful. If the indirect method is used, one can remove the Lagrange multiplier from the equality constrained problem and obtain a simple equation to solve. Using the indirect method results in a completely division-free algorithm, while the direct method has one division; the factorization of the KKT matrix prior to starting.

The termination criteria for the algorithm is that the norms of the primal and dual residuals are smaller than some tolerance level $\epsilon_{prim} > 0$ and $\epsilon_{dual} > 0$. These are set to

$$\begin{aligned} \epsilon_{prim} &= \epsilon_{abs} + \epsilon_{rel} \max\{\|Ax^k\|_\infty, \|z^k\|_\infty\} \ , \\ \epsilon_{dual} &= \epsilon_{abs} + \epsilon_{rel} \max\{\|A^\top y^k\|_\infty, \|q\|_\infty\} \ , \end{aligned} \tag{4.2}$$

where the value of ϵ_{abs} can be tweaked by the designer.

Operator splitting methods typically return average accuracy. This algorithm performs solution polishing after the ADMM terminates to return a high accuracy solution, which is done by making assumptions about which constraints are active in the solution.

A shortcoming of this algorithm that is worth mentioning, is that the convergence time is dependent on the choice of the step-size and relaxation parameters, σ, ρ and α. This choice is still an open research question, but the OSQP default parameters have been chosen and tuned by testing millions of problems with a wide range of dimensions [10].

Since this algorithm in particular can solve optimization problems to a relative degree of certainty while still using few iterations, therefore being computationally inexpensive, it has been deemed a practical use for embedded processors.

Figure 4.1, taken from Stellato et al. [27], displays the computation time of

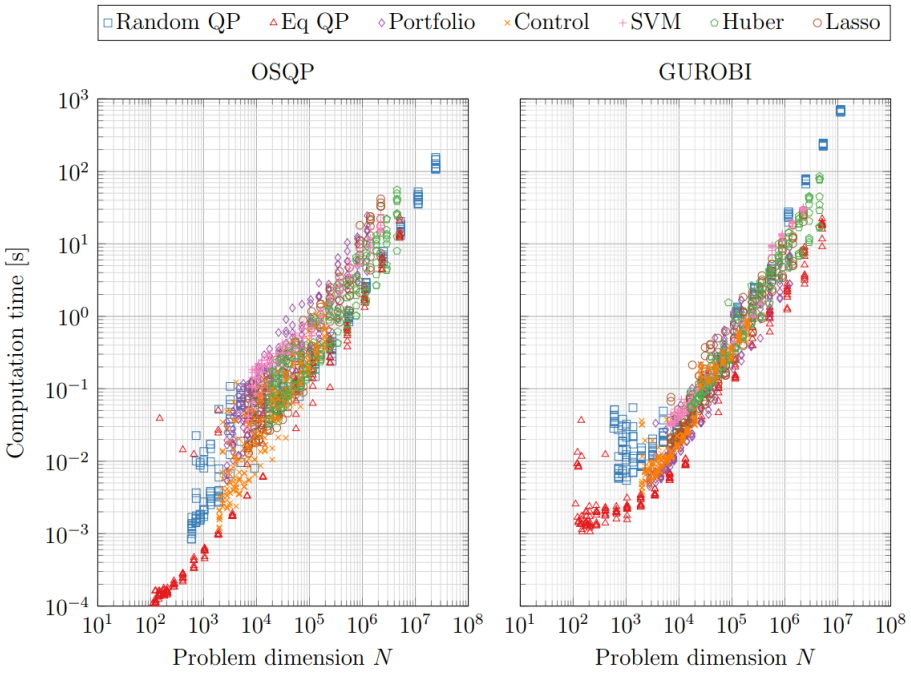

Figure 4.1: Computation time versus problem dimension for OSQP and GUROBI for 7 benchmark problem classes [27]

the OSQP solver and the GUROBI solver for 7 benchmark problems, and clearly demonstrates OSQP has a clear advantage in most of the problem. Specifically, the optimal control problem which has the same characteristics as the online problem for MPC, performs slightly better with OSQP.

Chapter 5

Developing a process model

In this section a state-space representation for the helicopter will be presented. First, a nonlinear model of the helicopter will be derived, with the three degrees of freedom as states and the actuator voltages as inputs. This model will then be linearized, and a pitch and elevation controller will be introduced so the new control inputs become the pitch reference p_c and the elevation reference e_c.

5.1 Helicopter model

Nonlinear model.

A schematic figure of the helicopter, with the appropriate definitions, can be seen in Figure 5.1.

It is assumed that the the forces generated by the propellers is proportional to the voltages applied,

$$\begin{aligned} F_f &= K_f V_f \ , \\ F_b &= K_b V_b \ , \end{aligned} \tag{5.1}$$

where K_f and K_b are the proportionality constants and are assumed to be identical, ($K_f = K_b$). According to Newton's third law of motion,

$$F_{g,f} + F_{g,b} = F_f + F_b = K_f V_s^* \ , \tag{5.2}$$

where the voltage sum $V_s^* = V_b + V_f$ is the voltage sum needed to keep the helicopter in equilibrium. From here, the expression for the motor force

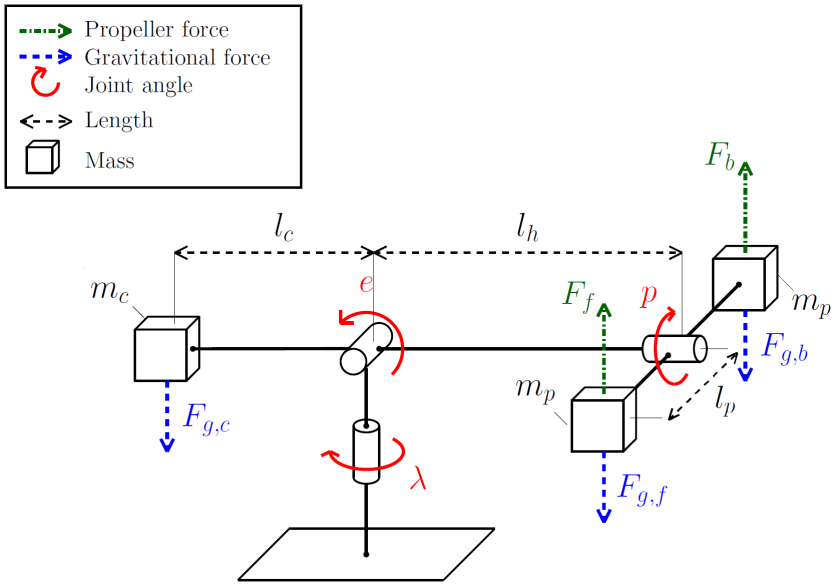

Figure 5.1: Diagram of helicopter forces, angles and physical sizes

constant is easily calculated as

$$K_f = \frac{m_g \cdot g}{V_s^*} \ , \tag{5.3}$$

where g is the gravitational acceleration and m_g is the effective mass of the helicopter.

Moving on to the equations for the helicopter states, Newton's second law of rotation regrading the rotation about a single principal axis, states that the net external torque is equal to the product of the moment of inertia and the angular acceleration. The equations of motions will be stated for one angle at a time, starting with the travel angle λ.

The net external torque of travel is

$$\sum \tau_\lambda = (F_b - F_f) l_h \cos e \sin p \ , \tag{5.4}$$

so the angular acceleration for travel can be expressed as

$$\ddot{\lambda} = -\frac{K_f l_h V_d}{J_\lambda} \cos e \sin p \ , \tag{5.5}$$

where the voltage difference $V_d = V_f - V_b$ and the moment of inertia for travel is $J_\lambda = 4 m_p l_h^2$.

Moving on to the pitch angle p the contributions are the propellers force, such that

$$\sum \tau_p = (F_f - F_b) l_p \ . \tag{5.6}$$

This results in the pitch angular acceleration being

$$\ddot{p} = \frac{K_f l_p}{J_p} V_d \ , \tag{5.7}$$

with the moment of inertia for pitch being $J_p = 2 m_p l_p^2$.

For elevation e, the contributions are the propellers affected by the angle of the pitch as well as the weight of both arms, so

$$\sum \tau_e = (F_f + F_b) l_h \cos p - 2 m_p g l_h \cos e + m_c g l_c \cos e \ . \tag{5.8}$$

The angular acceleration for elevation will be

$$\ddot{e} = \frac{1}{J_e}((m_c l_c - 2 m_p l_h) g \cos e + K_f l_h V_s \cos p) \ , \tag{5.9}$$

where the moment of inertia for elevation is $J_e = 4 m_p l_h^2$.

Note that all friction, both aerodynamic drag and friction in the joints, has been neglected.

By defining the constants,

$$\begin{aligned} L_1 &= \frac{K_f l_h}{J_\lambda} \ , \\ L_2 &= \frac{K_f l_p}{J_p} \ , \\ L_3 &= \frac{K_f l_h}{J_e} \ , \\ L_4 &= \frac{g(m_c l_c - 2 m_p l_h)}{J_e} \ , \end{aligned} \tag{5.10}$$

the system equations can be written,

$$\begin{aligned} \ddot{\lambda} &= L_1 V_d \cos e \sin p \ , \\ \ddot{p} &= L_2 V_d \ , \\ \ddot{e} &= L_3 V_s \cos p + L_4 \cos e \ . \end{aligned} \tag{5.11}$$

Linearization.

Since the MPC solves a QP problem at every time step, the system equations in (5.11) will be linearized around the helicopters equilibrium point.

The linear approximation of the function $y = f(x, u)$ about (x^*, u^*) using the first-order Taylor series and defining the new states $\Delta x = (x - x^*), \Delta u = (u - u^*)$ is given,

$$y = f(x, u) \approx f(x^*, u^*) + \frac{\partial f}{\partial x}|_{(x^*, u^*)} \Delta x + \frac{\partial f}{\partial u}|_{(x^*, u^*)} \Delta u \ . \tag{5.12}$$

Given the nonlinear equations of motions of the helicopter system presented in Equation (5.11), the linear approximation in Equation (5.12) can be used to obtain a linear approximation.

In addition, the resulting linear system should be a first-order system, so let the states be

$$x^\top = \begin{bmatrix} \lambda & \dot{\lambda} & p & \dot{p} & e & \dot{e} \end{bmatrix} \ , \tag{5.13}$$

while the inputs are

$$u^\top = \begin{bmatrix} V_s & V_d \end{bmatrix} \ . \tag{5.14}$$

This results in,

$$\dot{x} = f(x, u) = \begin{bmatrix} \dot{\lambda} \\ L_1 V_s \cos e \sin p \\ \dot{p} \\ L_2 V_d \\ \dot{e} \\ L_3 V_s \cos p + L_4 \cos e \end{bmatrix} \ . \tag{5.15}$$

The equilibrium point of the system $x^* = 0$ results in the equilibrium inputs $V_s^* = -\frac{L_4}{L_3}, V_d^* = 0$. The resulting linearization becomes

$$\begin{aligned} &\dot{x} = Ax + Bu \ , \\ &A = \begin{bmatrix} 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & K_2 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \ B = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & K_1 \\ 0 & 0 \\ K_3 & 0 \end{bmatrix} \ , \end{aligned} \tag{5.16}$$

with

$$\begin{aligned} K_1 &= L_2 \ , \\ K_2 &= \frac{L_1 L_4}{L_3} \ , \\ K_3 &= L_3 \ . \end{aligned} \tag{5.17}$$

The incremental encoders on the helicopter supply the travel angle, pitch angle, elevation angle to a Simulink model that calculates the travel rate, pitch rate and elevation rate. Therefore the output equation becomes

$$\begin{aligned} y &= Cx + Du \ , \\ C &= \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}, \quad D = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 0 \end{bmatrix} \end{aligned} \tag{5.18}$$

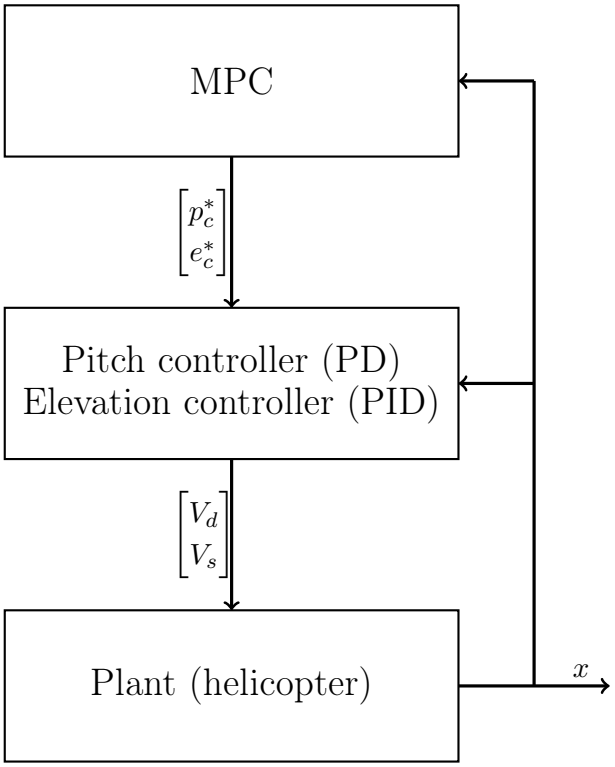

Figure 5.2: Diagram of system control architecture

Lower level of control.

When using model predictive control, it is usually part of a multi-level hierarchy of control functions, where the highest level is the MPC that returns reference values for some lower level of control. For this reason, a pitch controller and elevation controller is implemented, and the new control inputs for the process model used in MPC will be $u^\top = \begin{bmatrix} p_c & e_c \end{bmatrix}$. Figure 5.2 illustrates the architecture of this control hierarchy.

The controllers for the pitch and elevation are

$$\begin{aligned} V_d &= K_{pp}(p_c - p) - K_{pd}\dot{p} \ , \\ V_s &= K_{ep}(e_c - e) - K_{ed}\dot{e} \ , \end{aligned} \tag{5.19}$$

where the controller coefficients have been selected by choosing a bandwidth w_p, w_e and relative damping frequency d_p, d_e for each controller. These values are found in Appendix A.

These are added to the linearized state space in (5.16), and the resulting *discrete* state-space representation of the helicopter is,

$$
\begin{aligned}
x^+ &= Ax + Bu\,, \\
A &= \begin{bmatrix}
1 & h & 0 & 0 & 0 & 0 \\
0 & 1 & -hK_2 & 0 & 0 & 0 \\
0 & 0 & 1 & h & 0 & 0 \\
0 & 0 & -hK_1K_{pp} & 1 - hK_1K_{pd} & 0 & 0 \\
0 & 0 & 0 & 0 & 1 & h \\
0 & 0 & 0 & 0 & -hK_3K_{ep} & 1 - hK_3K_{ed}
\end{bmatrix}, \\
B &= \begin{bmatrix}
0 & 0 \\
0 & 0 \\
0 & 0 \\
hK_1K_{pp} & 0 \\
0 & 0 \\
0 & hK_3K_{ep}
\end{bmatrix},
\end{aligned}
\tag{5.20}
$$

with $x = \begin{bmatrix} \lambda & \dot{\lambda} & p & \dot{p} & e & \dot{e} \end{bmatrix}^\top$, $u = \begin{bmatrix} p_c & e_c \end{bmatrix}^\top$.

Though a PID controller is implemented for the control of elevation, equation (5.19) only implements a PD controller for the elevation. Because a PID controller means augmenting the system to add an integral state, a different approach is taken. Consider the equation for the elevation acceleration

$$\ddot{e} = L_3 V_s \cos p + L_4 \cos e \ , \tag{5.21}$$

which was linearized to

$$\ddot{e} = L_3 V_s. \tag{5.22}$$

The term $L_4 \cos e$, when the helicopter is around the equilibrium point, which is the main operating region, can be expressed as

$$L_4 \cos e \approx L_4 \ . \tag{5.23}$$

This will cause a constant deviation in the elevation if left ignored. For this reason an integral term is added to the implementation of the elevation controller, to remove this constant, and these terms are then considered to cancel each other out, which is why they do not appear in the model equations.

5.2 Selection of constraints

It was necessary to add constraints on the travel λ, pitch p and pitch reference p_c.

The constraints on the travel, λ, is there to ensure feasibility. Starting too far away from the terminal set $\mathbb{X}_f$ leads to infeasibility, because the controller cannot achieve a state trajectory that ends in the set in N steps. Therefore a hard constraint on λ is only there to avoid infeasibility during run-time.

The pitch constraint is a so-called *soft constraint* because slack variables were implemented for this constraint. The reason for the pitch reference having a tighter constraint than the pitch is because of possible disturbances in the actuators. There needs to be a margin in case the pitch reference gives a larger pitch than the constraint, and the next iteration of the problem becomes infeasible.

The constraints are

$$-c_\lambda \leq \lambda \leq c_\lambda \ , \tag{5.24}$$

$$-(c_p + \epsilon_p) \leq p \leq (c_p + \epsilon_p) \ , \tag{5.25}$$

$$-c_{p_c} \leq p_c \leq c_{p_c} \ , \tag{5.26}$$

with the values

$$\begin{aligned} c_\lambda &= 3.665 \text{ rad } (210 \text{ deg}) \ , \\ c_p &= 0.524 \text{ rad } (30 \text{ deg}) \ , \\ c_{p_c} &= 0.436 \text{ rad } (25 \text{ deg}) \ . \end{aligned} \tag{5.27}$$

5.3 Modeling the constant disturbance

There is a physical disturbance in the system that is very easy both to notice and to model.

The disturbance comes from the fact that the propellers in the helicopter spin in the same direction, thus creating a torque moment around the travel angle. This leads to a constant drift in the travel, which will be referred

to as d. Given the description of this constant disturbance, the following augmented state-space is presented,

$$\begin{aligned} \begin{bmatrix} x^+ \\ d^+ \end{bmatrix} &= \begin{bmatrix} A & A_d \\ \mathbf{0} & 1 \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + \begin{bmatrix} B \\ \mathbf{0} \end{bmatrix} u \ , \\ y &= \begin{bmatrix} C & C_d \end{bmatrix} \begin{bmatrix} x \\ d \end{bmatrix} + Du \ , \end{aligned} \tag{5.28}$$

with

$$A_d = h \cdot \begin{bmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \ , C_d = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} \ . \tag{5.29}$$

Chapter 6

Hardware and Software Implementation

This chapter focuses on the implementation of an MPC controller on the helicopter, both the hardware and software aspect. The implementation of the algorithm described in Chapter 4 will be covered, along with the HIL architecture of the implementation. Then the implementation of the online optimal control problem in OSQP is covered. Furthermore, the Simulink model used is discussed in detail, and the C code used to call the QP solver is shown and explained.

6.1 OSQP

There is a lot of software available for solving QP problems, all suitable for different types of problem and different applications.

Numerous reviews have been done on the performance of QP solvers for embedded MPC, [4], [3]. Suitable alternatives to OSQP include qpOASES, and CVXGEN, where the latter is a code generation software for convex optimization. However, both of these are more than five years old, and therefore outdated.

The QP solver used in this work, was introduced in 2018 by Stellato et al. [27] for general purpose QP problems. The algorithm used is described in detail in Chapter 4, and has been implemented in the open-source Operator Splitting

Quadratic Program (OSQP) and is written in C with a very small code footprint. The implementation can be used in C, C++, Fortran, Python, Matlab, Julia and Rust.

In addition, the QP solver offers a software package in both MATLAB and Python that can generate C code tailored to a specific quadratic program [2]. This functionality is used in this project to generate C code designed specifically for the online optimization problem presented in Chapter 3, and can then be updated at every time step, using the C interface.

The fact that this solver has both a MATLAB and C code interface and a code generation software package, made it perfect to use in this project, where both running a simulation in MATLAB and using C code in the Simulink model was important.

One downside of this implementation of OSQP is that the code generation software package enables an embedded flag which then disables the profiling function. The profiling gives the user the ability to retrieve the solve time, set up time and update time for the OSQP function. To work around this and be able to see the solve time of the OSQP, a timer was implemented separately.

6.2 Hardware-in-the-loop (HIL)

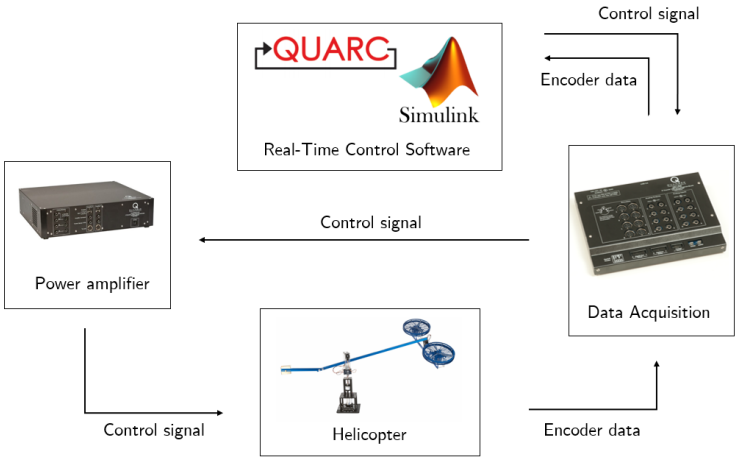

Figure 6.1: Information flow in Hardware-in-the-loop (HIL) testing

Figure 6.1 shows a diagram of the Hardware-in-the-loop setup.

The hardware used to steer the helicopter is also supplied. It consists of the Q8-USB 8 Channel USB Data Acquisition Board and the VoltPAQ-X2 2 Channel Linear Voltage Amplifier from Quanser.

The board is an I/O card made for prototyping and Hardware-in-the-loop development, and receives the signals from the helicopter, which is equipped with sensors. The amplifier receives the system inputs from the written control software supplied by the user and applies voltage to the propellers.

In addition to the hardware described above, Quanser's 3 DOF helicopter package also comes with QUARC Real-Time Control Software for MATLAB and Simulink.

QUARC extends the code generation capabilities already existent in Real-Time Workshop (now called Simulink Coder), while adding new set of targets the can change the source code generated by Real-Time Workshop to suit the particular platform. QUARC then compiles and links this code with relevant libraries and downloads the code onto the target, which is the Q8 Board. QUARC also has External Mode Communications that allows for connection to the target in Simulink for receiving the signals in real time. To enable this, QUARC Targets Library has a whole library of Simulink blocks that add a lot of HIL functionality, that allows one to use the signals from the board in the Simulink model.

The Simulink blocks written by QUARC that are used in the Simulink model are

- HIL Initialize
- HIL Read Encoder Timebase
- HIL Write Analog

These are used for initializing the HIL functionality, reading encoder information from the helicopter so that it can be utilized in the Simulink model and writing voltage data to the IO card.

When dealing with a hierarchical control architecture, where the highest level is MPC and the lower levels are the pitch and elevation controller, the highest level should have a lower frequency. This is so that the lower level controllers can have time to reach the given reference value before a new one is calculated.

QUARC fully supports multithreading [20], blocks running on different sampling intervals are simply split into multiple threads, grouped together by sampling rate. In this simulation, the lower levels of control run at 50 Hz, while the MPC runs at 12.5 Hz, and is also tested at 25 Hz and 5 Hz.

6.3 Implementing the optimization problem for OSQP

The optimization problem solved in OSQP has the form,

$$\begin{aligned} \min_{\phi} \quad & \frac{1}{2}\phi^\top P\phi + q^\top \phi \\ \text{subject to} \quad & lb \leq A\phi \leq ub \ . \end{aligned} \tag{6.1}$$

Let the objective function

$$V(\mathbf{x},\mathbf{u}) = \frac{1}{2}\sum_{j=0}^{N-1}(x_n^\top Q x_n + u_n^\top R u_n) + x_N^\top P x_N \ , \tag{6.2}$$

be expressed as

$$\mathbf{x}^\top \tilde{Q}\mathbf{x} + \mathbf{u}^\top \tilde{R}\mathbf{u} \ , \tag{6.3}$$

where

$$\tilde{Q} = \begin{bmatrix} Q & & & & \\ & Q & & & \\ & & \ddots & & \\ & & & Q & \\ & & & & P \end{bmatrix}, \ \tilde{R} = \begin{bmatrix} R & & & \\ & R & & \\ & & \ddots & \\ & & & R \end{bmatrix} . \tag{6.4}$$

Inserting equation (3.13) ($\mathbf{x} = S_u\mathbf{u} + S_x x_0$) results in

$$\begin{aligned} V(\mathbf{u}) &= \mathbf{u}^\top(S_u^\top \tilde{Q} S_u + \tilde{R})\mathbf{u} + 2x_0^\top S_x^\top \tilde{Q} S_u \mathbf{u} + x_0^\top S_x^\top \tilde{Q} S_x x_0 \\ &= \mathbf{u}^\top H \mathbf{u} + x_0^\top f \mathbf{u} + c \ , \end{aligned} \tag{6.5}$$

with

$$H = S_u^\top \tilde{Q} S_u + \tilde{R}, \ \ f = 2S_x^\top \tilde{Q} S_u \ , \tag{6.6}$$

and the constant term c being neglected, since it does not affect the optimal solution.

The term for the cost of the slack variables,

$$\epsilon_{\mathbf{p}}{}^{\top} \tilde{S} \epsilon_{\mathbf{p}} \ , \tag{6.7}$$

has to be added as well, where

$$\tilde{S} = \begin{bmatrix} S & & \\ & \ddots & \\ & & S \end{bmatrix} , \epsilon_p = \begin{bmatrix} \epsilon_{p0} \\ \vdots \\ \epsilon_{pN} \end{bmatrix} \ . \tag{6.8}$$

The objective function in equation (6.1) can now be expressed as

$$\frac{1}{2} \begin{bmatrix} \mathbf{u} & \epsilon_{\mathbf{p}} \end{bmatrix} \cdot 2 \cdot \begin{bmatrix} H & 0 \\ 0 & \tilde{S} \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \epsilon_{\mathbf{p}} \end{bmatrix} + \begin{bmatrix} f & 0 \end{bmatrix} \cdot \begin{bmatrix} \mathbf{u} \\ \epsilon_{\mathbf{p}} \end{bmatrix} \ . \tag{6.9}$$

Let the constraints $x \in \mathbb{X}, u \in \mathbb{U}$ be written as

$$\begin{aligned} Cx + Du + E\epsilon_p &\leq e \ , \\ Gx_n &\leq h \ , \\ C, G \in \mathbb{R}^{a \times n}, \ D &\in \mathbb{R}^{a \times m} \ , \end{aligned} \tag{6.10}$$

with a being the total number of constraints.

Let the constraints for the whole horizon be

$$\tilde{C}\mathbf{x} + \tilde{D}\mathbf{u} + \tilde{E}\epsilon_{\mathbf{p}} \leq \tilde{e} \ , \tag{6.11}$$

with

$$\tilde{C} = \begin{bmatrix} C & & \\ & \ddots & \\ & & C \\ & & G \end{bmatrix} , \tilde{D} = \begin{bmatrix} D & & \\ & \ddots & \\ & & D \\ 0 & \dots & 0 \end{bmatrix} , \tilde{E} = \begin{bmatrix} E & & \\ & \ddots & \\ & & E \\ 0 & \dots & 0 \end{bmatrix} , \tilde{e} = \begin{bmatrix} e \\ \vdots \\ e \\ h \end{bmatrix} \ , \tag{6.12}$$

and

$$\mathbf{x} = \begin{bmatrix} x_1 \\ x_1 \\ \dots \\ x_{N-1} \end{bmatrix} , \mathbf{u} = \begin{bmatrix} u_0 \\ u_1 \\ \dots \\ u_N \end{bmatrix} , \epsilon_{\mathbf{p}} = \begin{bmatrix} \epsilon_{p0} \\ \epsilon_{p1} \\ \dots \\ \epsilon_{pN} \end{bmatrix} \ . \tag{6.13}$$

Again, inserting equation (3.13) into equation (6.11) to remove the state trajectory, gives

$$(\tilde{C}S_u + \tilde{D})\mathbf{u} + \tilde{E}\epsilon_{\mathbf{p}} \leq \tilde{e} - \tilde{C}S_x x_0 \ . \tag{6.14}$$

Since x_0 is a variable, in the sense that it changes for every computation of the optimization problem, the constraints are split up as follows,

$$A\phi \leq b_{in} + c_{in} \cdot x_0 \ , \tag{6.15}$$

where

$$A = \begin{bmatrix} \tilde{C}S_u + \tilde{D} & \tilde{E} \end{bmatrix} \ , b_{in} = \tilde{e} \ , c_{in} = -\tilde{C}S_x \ . \tag{6.16}$$

The lower bound lb will only have infinite terms.

The terms G, h and P can easily be removed when the terminal constraint and cost are not a part of the problem.

Since the upper bound $ub = b_{in} + c_{in} \cdot x_0$ and the linear cost $q = x_0^\top f$ both vary with the 'initial state' x_0, the current state of the system, when the online problem is solved, these arrays will have to be updated before every call to OSQP.

The MATLAB script that generates the matrices presented here can be found in appendix B, in listing B.4.

The terminal set for this system,

$$\mathbb{X}_f = \{x \mid Gx \leq h\} \tag{6.17}$$

is in six dimensions so a visualization of the whole set is not possible. However, the terminal set has been 'sliced' through the first and second dimensions, so the new sliced set is

$$\mathbb{X}_{f_{1:2}} = \{x \ \mid \ [G^{\text{col}_1} \ G^{\text{col}_2}]\, x \leq h\} \ . \tag{6.18}$$

This attempts to illustrate the constraints of two states in two dimensions, when the other states are zero. This is shown in figure 6.2.

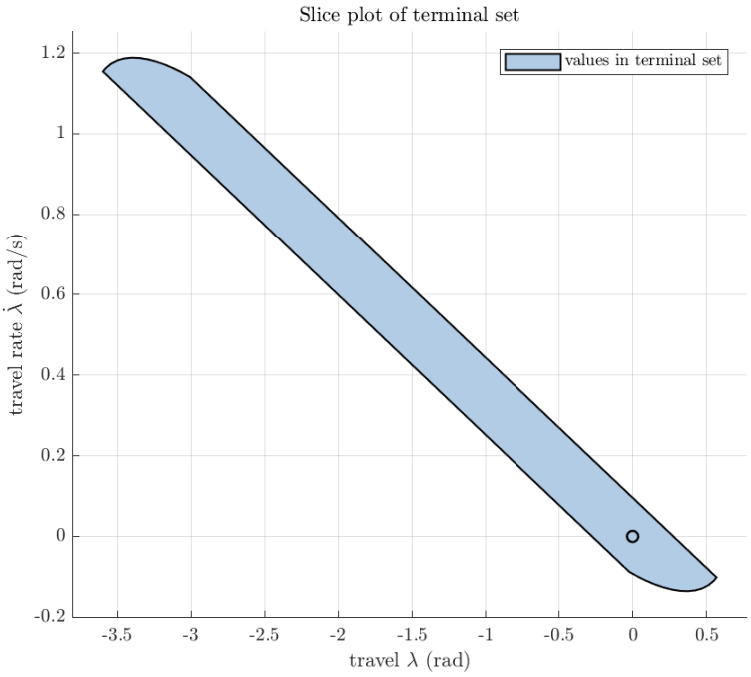

Figure 6.2: A sliced plot of the terminal set $\mathbb{X}_f$

6.4 Simulink Model

The basis for the Simulink model is a model created by Bjarne Anton Foss, Petter Tøndel, og Geir Stian Landsverk for the original version of the lab in TTK4135.

The full model is shown in Figure 6.3. The part of the model that was not developed during this project[1] can be found in Appendix C and will not be discussed here.

Notice the sums in the travel and elevation signal, coming out of the helicopter interface. These have been added because the goal for the control of the helicopter was to steer it from one position in travel and elevation, $(\lambda, e) = (\lambda^0, e^0)$ to the equilibrium point $(\lambda, e) = (0, 0)$. The encoder values from the helicopter are set to zero every time Simulink is connected, therefore constants have to be added to the signal so at start-up, it evaluates its current position as $(\lambda, e) = (\lambda^0, e^0)$. The MPC and the estimator will be discussed

[1] the pitch controller (aqua), elevation controller (light blue), and the voltage conversion and helicopter interface (green)

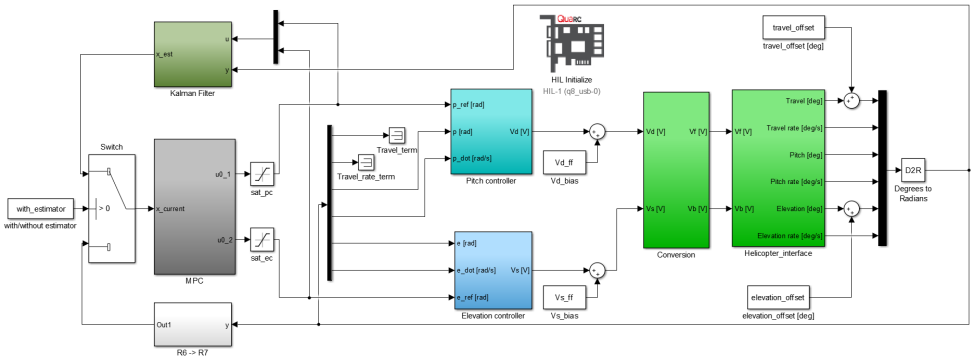

Figure 6.3: Complete Simulink model

in detail. Additionally, the switch block allows the user to easily turn off and on the estimator functionality, with the flag `with_estimator`.

The saturation blocks on the control inputs have been added as well. This is because when OSQP detects an infeasible problem, it returns the maximum value for the given data type as the solution. In MATLAB, this is `NaN` and in C it is '$2 \cdot 10^9$'. The saturation blocks are added to limit the actual input applied to the helicopter, as to not damage the equipment. The saturation is from -1 to 1.

6.4.1 MPC

First, the MPC was implemented, shown in Figure 6.4.

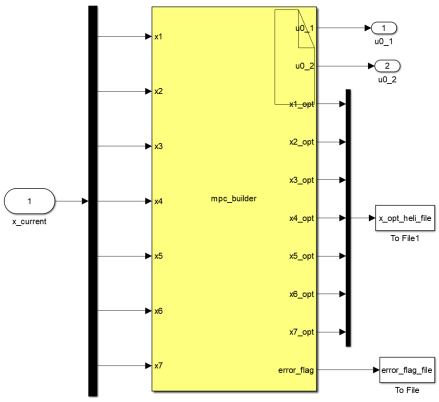

Figure 6.4: Simulink model of MPC block

This block is an S-Function Builder which serves as a wrapper for a generated

S-Function in the model. When this block is called, Simulink invokes a generated S-function that calls the C code provided [16]. The C code that is called is shown in Listing 6.1.

The new linear cost `q_new` and upper bound `ub_new` arrays are defined and updated based on the input to the block, which is the current state of the system, (`x1 x2 x3 x4 x5 x6 x7`). Then the `workspace`-structure is updated with these new arrays, in lines 35 and 36, and the problem is solved in line 40 and in line 44. An array for the optimal state trajectory is defined, and then using equation (3.13), the optimal state trajectory is obtained.

Listing 6.1: C code in the S-Function Builder block

```
1     //initialization
2     PyTimer * timer;
3     timer = malloc(sizeof(PyTimer));
4     c_float q_new[LENGTH_Z];
5     c_float ub_new[NUM_CONSTRAINTS];
6     int i;
7     int j;
8
9     //calculate new linear cost
10    for (i = 0; i < LENGTH_Z; i++)
11    {
12        q_new[i] = *x1 * fdata[0][i];
13        q_new[i] += *x2 * fdata[1][i];
14        q_new[i] += *x3 * fdata[2][i];
15        q_new[i] += *x4 * fdata[3][i];
16        q_new[i] += *x5 * fdata[4][i];
17        q_new[i] += *x6 * fdata[5][i];
18        q_new[i] += *x7 * fdata[6][i];
19    }
20
21    //calculate new upper bound
22    for (i = 0; i < NUM_CONSTRAINTS; i++)
23    {
24        ub_new[i] = bindata[i];
25        ub_new[i] += cindata[i][0] * *x1;
26        ub_new[i] += cindata[i][1] * *x2;
27        ub_new[i] += cindata[i][2] * *x3;
28        ub_new[i] += cindata[i][3] * *x4;
29        ub_new[i] += cindata[i][4] * *x5;
30        ub_new[i] += cindata[i][5] * *x6;
```

```
31          ub_new[i] += cindata[i][6] * *x7;
32      }
33
34      // update qp problem
35      *error_flag_q = osqp_update_lin_cost(&workspace, q_new);
36      *error_flag_ub = osqp_update_upper_bound(&workspace,
            ub_new);
37
38      //solve
39      tic(timer);
40      osqp_solve(&workspace);
41      *solve_time = toc(timer);
42
43      //calculate optimal state x = Su * u + Sx * x0;
44     c_float x[LENGTH_X];
45     for (i = 0; i < LENGTH_X; i++)
46     {
47         x[i] = Sxdata[i][0] * *x1;
48         x[i] += Sxdata[i][1] * *x2;
49         x[i] += Sxdata[i][2] * *x3;
50         x[i] += Sxdata[i][3] * *x4;
51         x[i] += Sxdata[i][4] * *x5;
52         x[i] += Sxdata[i][5] * *x6;
53         x[i] += Sxdata[i][6] * *x7;
54
55         for (j = 0; j < LENGTH_U; j++)
56         {
57             x[i] += Sudata[i][j] * ((&workspace)->solution->x[
                    j]);
58         }
59     }
60      *u0_1 = ((&workspace)->solution->x[0]);
61      *u0_2 = ((&workspace)->solution->x[1]);
62      *x1_1_opt = x[0];
63      *x1_2_opt = x[1];
64      *x1_3_opt = x[2];
65      *x1_4_opt = x[3];
66      *x1_5_opt = x[4];
67      *x1_6_opt = x[5];
68      *x1_7_opt = x[6];
```

Note that before and after the solve-function is called, a timer is started and

stopped, with the call to functions `tic()` and `toc()`. These functions are defined in Listing 6.2, and was taken from one of the files in the the code generation software package for OSQP, called 'emosqp_mex.c'.

Listing 6.2: timer.h

```
1  #include <windows.h>
2  #include <string.h>
3
4  typedef struct {
5      LARGE_INTEGER tic;
6      LARGE_INTEGER toc;
7      LARGE_INTEGER freq;
8  } PyTimer;
9
10 void tic(PyTimer* t)
11 {
12     QueryPerformanceFrequency(&t->freq);
13     QueryPerformanceCounter(&t->tic);
14 }
15
16 c_float toc(PyTimer* t)
17 {
18      QueryPerformanceCounter(&t->toc);
19
20     return ((t->toc.QuadPart - t->tic.QuadPart) / (c_float)t
           ->freq.QuadPart);
21 }
```

The functions retrieve a high resolution time stamp before and after the solve-function is called and terminated along with the frequency of the performance counter and returns the time in seconds between these two calls.

For the S-Function to be able to update the problem online, it needs access to the arrays b_{in}, c_{in} and f, and to obtain the optimal state trajectory it needs S_x and S_u. These are created in MATLAB and need to be defined in C and added to the 'workspace.h'-file, where the problem is defined.

This is done in a simple MATLAB script called 'gen_code_params', that generates strings that define the MATLAB arrays as multi-dimensional arrays in C. An exerpt of this is shown in Listing 6.3, lines 1 to 17. These are then copied into the 'workspace.h' file where the S-Function has access

to them. In addition the number of constraints a, the length of $\mathbf{x}$, $\mathbf{u}$ and the optimization variable ϕ are added as macros in lines 19 to 27, because they are susceptible to change if the number of constraints or optimization horizon is changed.

Listing 6.3: Exerpt from gen_code_params.m

```
1  %% cin
2
3  strng_cin = 'c_float cindata[';
4  strng_cin = strcat(strng_cin, num2str(size(cin,1), precision
       ));
5  strng_cin = strcat(strng_cin, '][');
6  strng_cin = strcat(strng_cin, num2str(size(cin,2), precision
       ));
7  strng_cin = strcat(strng_cin, '] = {');
8  for i = 1:size(cin,1)
9      strng_cin = strcat(strng_cin, '{');
10     for j = 1:size(cin,2)
11         strng_cin = strcat(strng_cin, '(c_float)');
12         strng_cin = strcat(strng_cin, num2str(cin(i,j),
               precision));
13         strng_cin = strcat(strng_cin, ',');
14     end
15     strng_cin = strcat(strng_cin, '} ,');
16 end
17 strng_cin = strcat(strng_cin, '};');
18
19 %% makros
20 makro_num_constr = '#define NUM_CONSTRAINTS';
21 makro_num_constr = [makro_num_constr ' ' num2str(size(Ain,
       1))];
22 makro_len_z = '#define LENGTH_Z';
23 makro_len_z = [makro_len_z ' ' num2str(size(Ain,2))];
24 makro_len_x = '#define LENGTH_X';
25 makro_len_x = [makro_len_x ' ' num2str(system.n * system.N)
       ];
26 makro_len_u = '#define LENGTH_U';
27 makro_len_u = [makro_len_u ' ' num2str(system.m * system.N)
       ];
```

The S-Function has certain limitations when used with code generation support. For instance, inputs and outputs cannot be Simulink buses, so

any change in the number of outputs and inputs must be changed manually.

6.4.2 Estimator

The other block developed during this project was the estimator, which is a simple Kalman filter and can be seen in Figure 6.5.

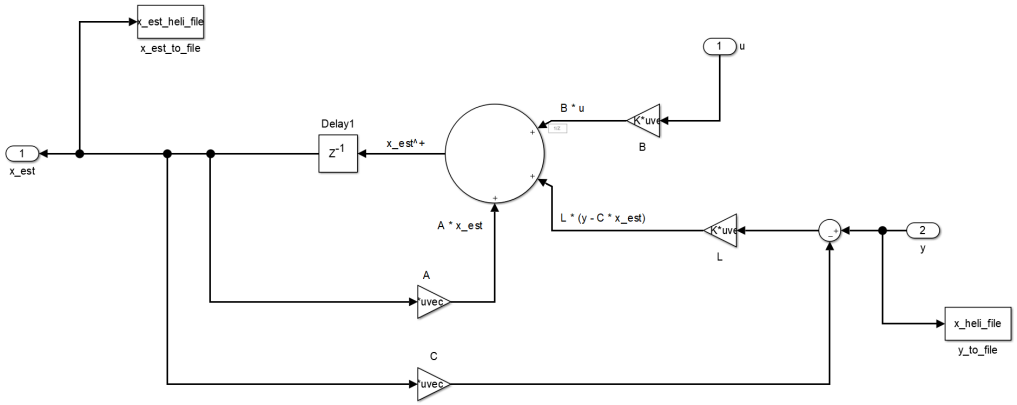

Figure 6.5: Simulink model of estimator

6.5 Multi-Parametric Toolbox

The software implementation, mainly the MATLAB code, also uses the Multi-Parametric Toolbox (MPT) 3.0 [11], which is an open-source MATLAB-based toolbox. This is used to create the structures `Polyhedron` and `LTIsystem`. These are used among other things to define the constraint set $\mathbb{X}$ and $\mathbb{U}$, and to calculate the terminal set $\mathbb{X}_f$.

The command `S = Polyhedron(A,B)` creates the Polyhedron-structure in MATLAB equivalent to the set $S = \{x \mid Ax \leq b\}$.

The command `model = LTISystem('A', A_K)` creates the structure `model` for the autonomous system given as

$$x^+ = A_K x \ . \tag{6.19}$$

The commands

```
model.x.with(setConstraint)
model.x.setConstraint = X
```

where `X` is a Polyhedron structure, sets the state constraints of the given system

$$x \in \mathbb{X} \ . \tag{6.20}$$

Lastly, `X_f = model.invariantSet();` returns the maximal positively invariant set for the system (6.19).

Chapter 7

Results

In this chapter, the results are presented. A numerical simulation in MATLAB of the controller applied to the system model with a simulated disturbance is shown in the first section, a long with a single plot showing the overall solve time of OSQP. In the next section, the result from the HIL experiments performed on the helicopter will be presented. Since the solve time of the OSQP is highly relevant when running on the actual hardware, more focus will be invested on this.

To make it easier to quantify and to compare the performance of the controller, all test runs attempt to move the system from a starting point in $(\lambda, e) = (-40, -30)$ to the equilibrium, and the performance is determined by how fast and accurate the control of the travel angle is. The initial condition for the elevation is due to the fact that the helicopter rests on a table when the Simulink is connected, at a 30 degree angle, so this is added to the signal so the elevation measurement is accurate for the relative coordinate system defined.

Due to the constant drift in travel, the pitch will have a constant deviation so that the travel can approach zero.

For simplicity, when the finite-horizon optimal control problem solved online contains a terminal cost and a terminal constraint, it will be referred to as *stable MPC*. Otherwise, it will be called *nominal MPC*.

7.1 Numerical simulation of helicopter model

The plots from the numerical simulation in MATLAB are presented in this section. The execution file for the simulation can be seen in Listing B.1 in Appendix B. The system is discretized with a sample time of $T_s = 0.08$ seconds and optimized over an optimization horizon $N = 15$, unless otherwise specified. The system is simulated with a generated disturbance sequence, containing random numbers between 3 and 3.6 degrees (0.052 and 0.063 radians).

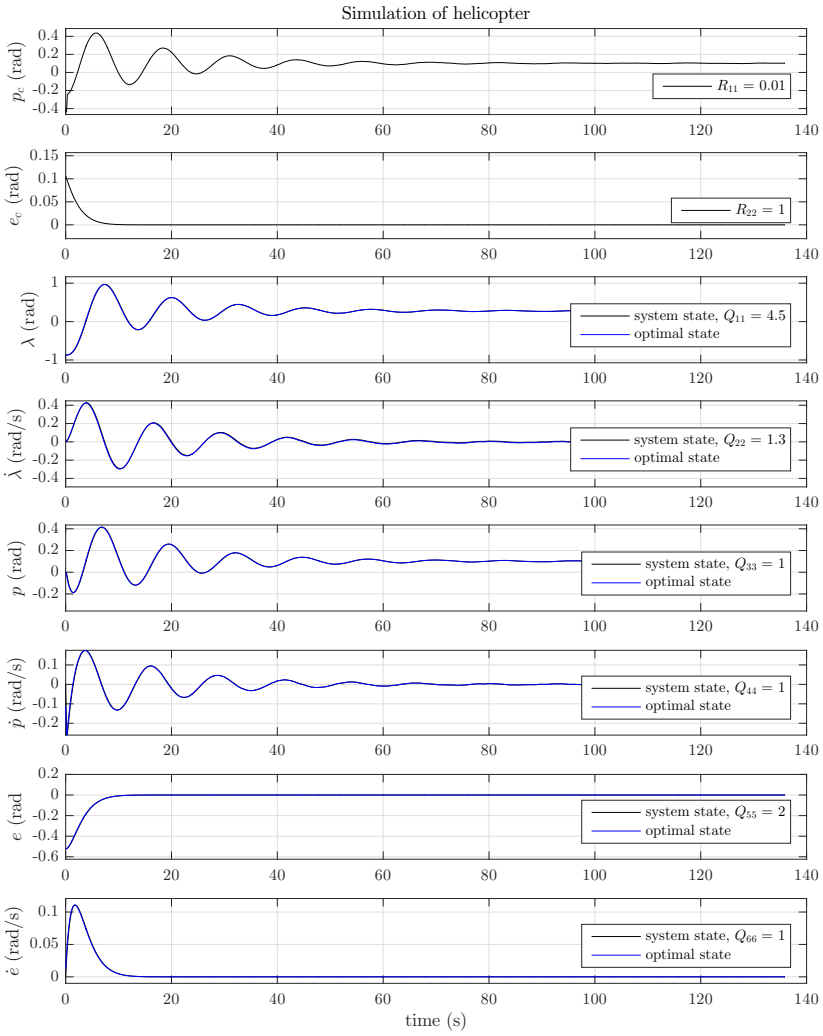

Figure 7.1: Simulation of nominal MPC

First, the simulation is controlled with nominal MPC in Figure 7.1. The travel settles at about 0.2761 radians, with a settling time of about 70 seconds. Notice the pitch also settles at a value above zero. This is proof enough that a constant disturbance is affecting the system, as the pitch needs to have a nonzero angle to keep the helicopter still.

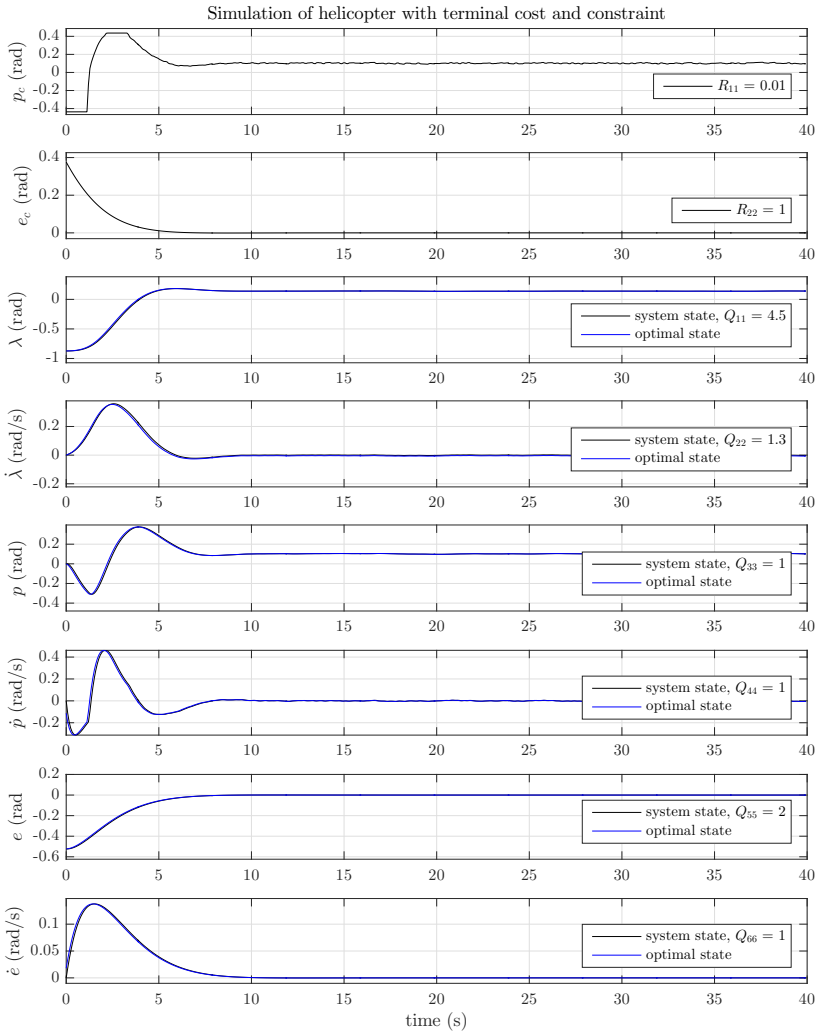

Figure 7.2: Simulation of stable MPC

Next, the terminal constraint and terminal cost is added to attempt to more quickly steer the system to stability, seen in Figure 7.2. The difference is quite clear. The settling time of the system is now about 10 seconds, and $\lambda = 0.1394$. Which means, the stability properties of the stable MPC also affects the constant deviation in travel.

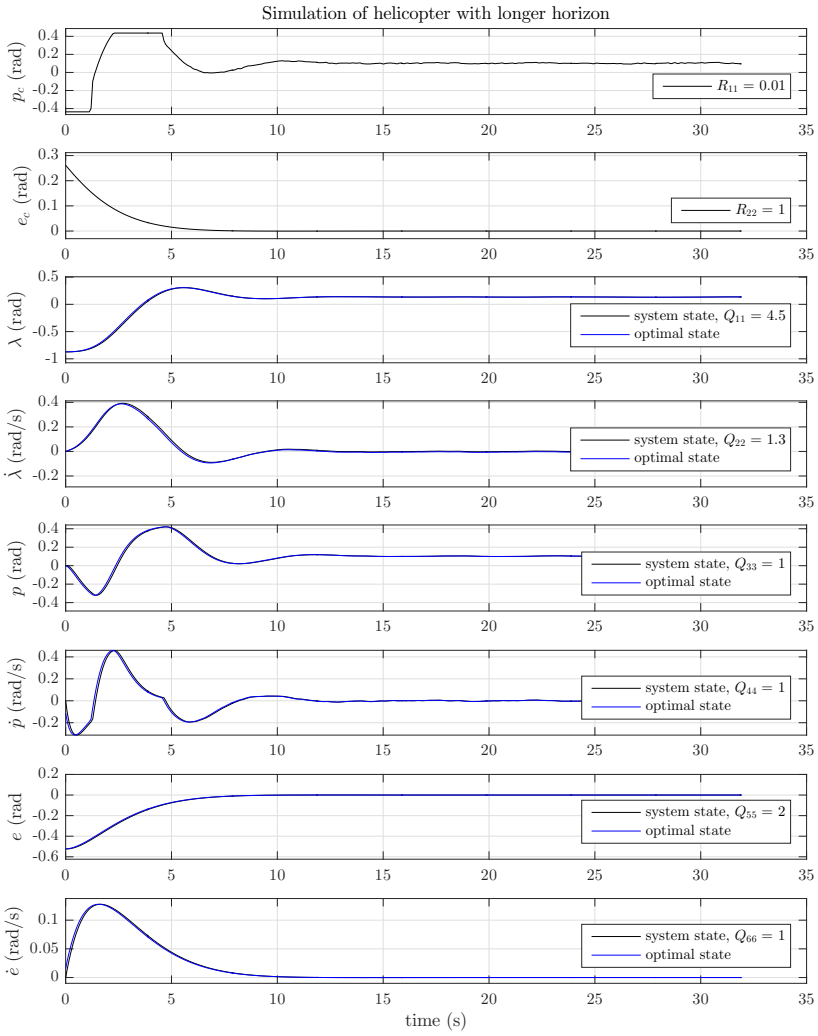

Figure 7.3: Simulation of nominal MPC with longer horizon

It was mentioned in the introduction of the terminal cost and constraint, that it is an attempt to emulate a longer optimization horizon without the computation expense that comes along with that. To illustrate this effect, the controller with double the optimization horizon, $N = 30$, has been simulated, seen in Figure 7.3. This control appears to have the same effect as the addition of the terminal terms. The travel settles at 0.1336 rad after about 12 seconds. However, the solve time of OSQP increases from 0.553 to 0.816

milliseconds because the longer optimization horizon leads to a large problem dimensions which involves more complex calculations online.

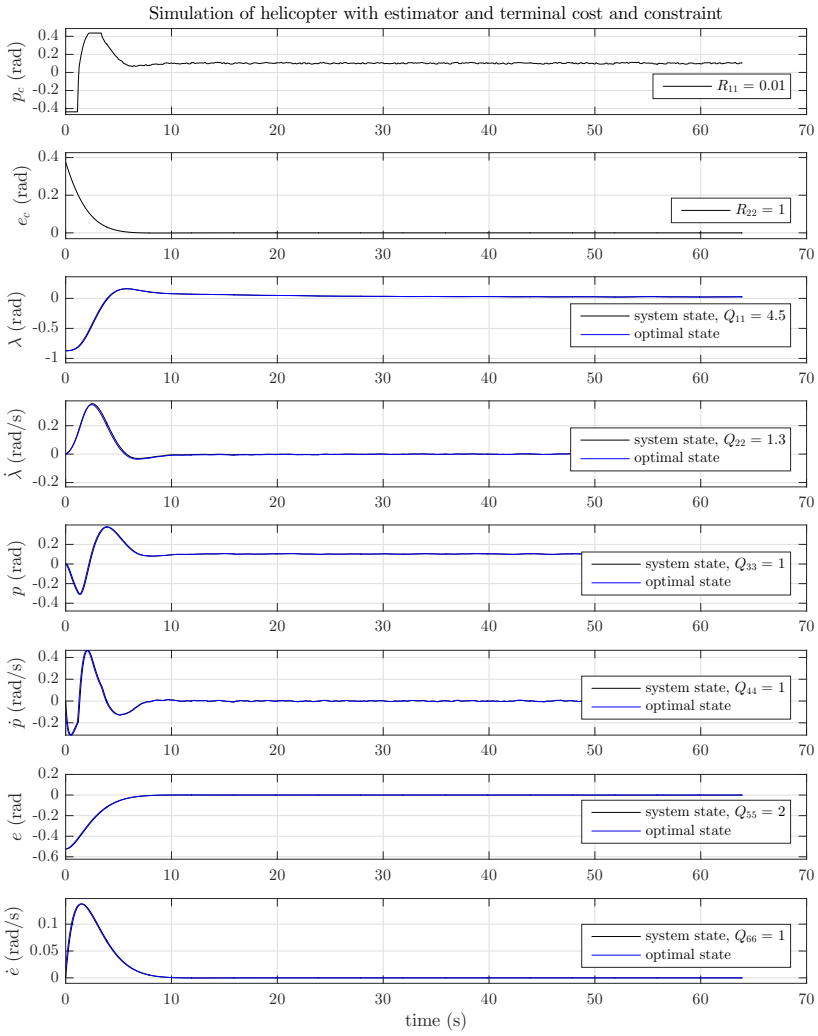

Figure 7.4: Simulation of stable MPC with Kalman filter

To remove the constant offset in travel, the helicopter model is augmented with an extra state to model the constant disturbance in travel and a Kalman filter is implemented to estimate this state so the controller can consider it when optimizing the control input. This is shown in Figure 7.4 to be successful. Here, λ settles on 0.01254 rad after about 12 seconds.

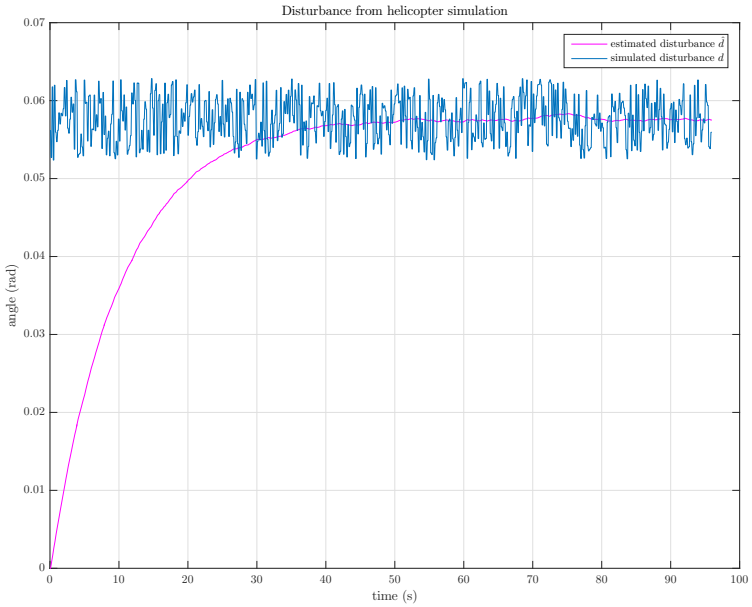

Figure 7.5: Estimation of disturbance from simulation

The estimation if the augmented state d from the Kalman filter was very similar for all simulations, both nominal and stable MPC, therefore only one example will be shown in Figure 7.5. This particular estimation is from the simulation described in Figure 7.4 of stable MPC. The estimated value in pink settles on about 0.06 radians, which as an accurate and smooth approximation of the generated disturbance sequence also displayed in the figure.

To attempt to quantify all the information obtained from running the numerical simulations, Table 7.1 summarizes the most important information from all the simulation results. The discussion chapter will refer to this table when analyzing the results.

Even though the choice of weights in the control problem usually has a significant effect on the performance, they remain unchanged in all the numerical simulations, so they will not be included in the table. To keep the focus on the results from the actual control of the helicopter, some figures have not been included although their information is included in the table, specifically the simulation and solve time of nominal MPC with a filter. These figures can be found in Appendix D.

Table 7.1: Results from numerical simulations

controller	horizon	settling time	lim λ	avg solve time
nominal	15	70 s	0.2761 rad	0.186 ms
nominal	30	12 s	0.1336 rad	0.816 ms
nominal with filter	15	90 s	0.2095 rad	0.196 ms
stable	15	10 s	0.1394 rad	0.553 ms
stable with filter	15	12 s	0.0254 rad	0.547 ms

7.2 Experimental results of helicopter performance

In this section, the results from the implementation of the controller being run on the physical helicopter will be presented. The optimization horizon is still $N = 15$ unless otherwise specified. The discretization interval and the sampling interval of the MPC is 0.08 seconds (12.5 Hz), while the rest of the model runs at 50 Hz.

Just like in the previous section, a table at the bottom of this section will summarize key information given in the figures.

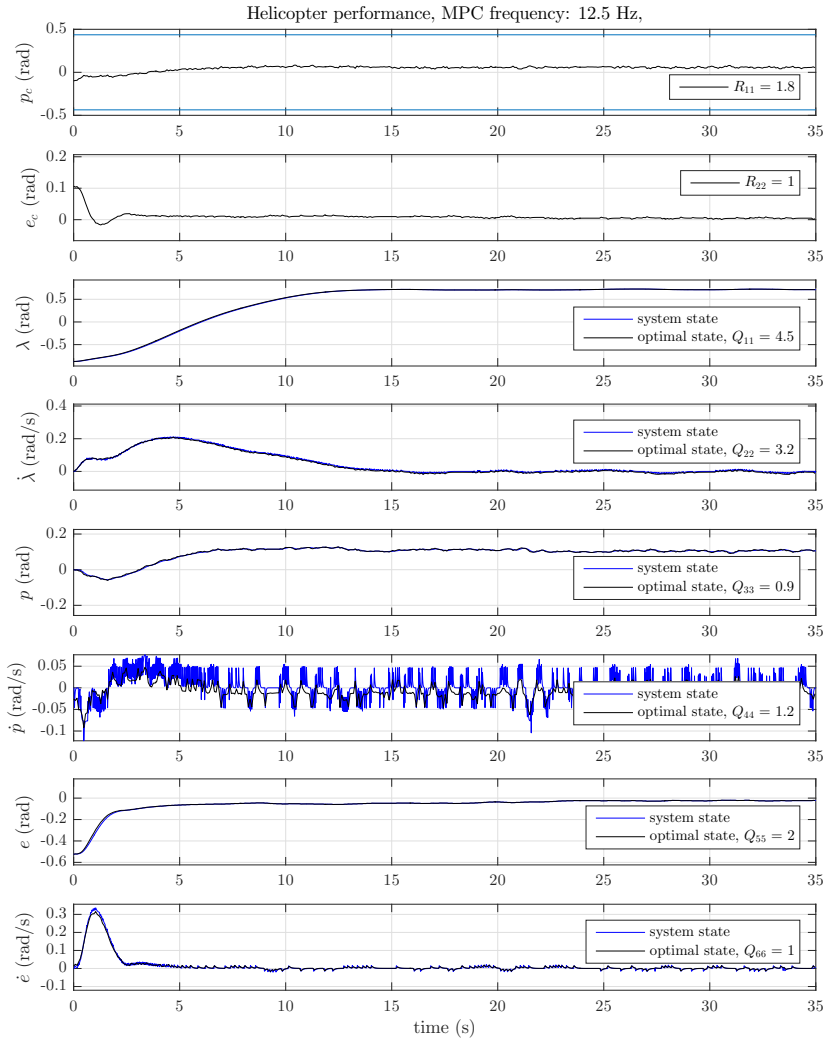

Figure 7.6: Online performance of nominal MPC

Figure 7.6 display the helicopter performance when controlled with nominal MPC. The value of λ approaches 0.719 rad after about 17 seconds. Notice the oscillations in the pitch rate $\dot{p}$, that are also somewhat present in the pitch p and pitch reference p_c as well. They are due to model inaccuracies caused by nonlinearities in the system, and will be discussed the next chapter.

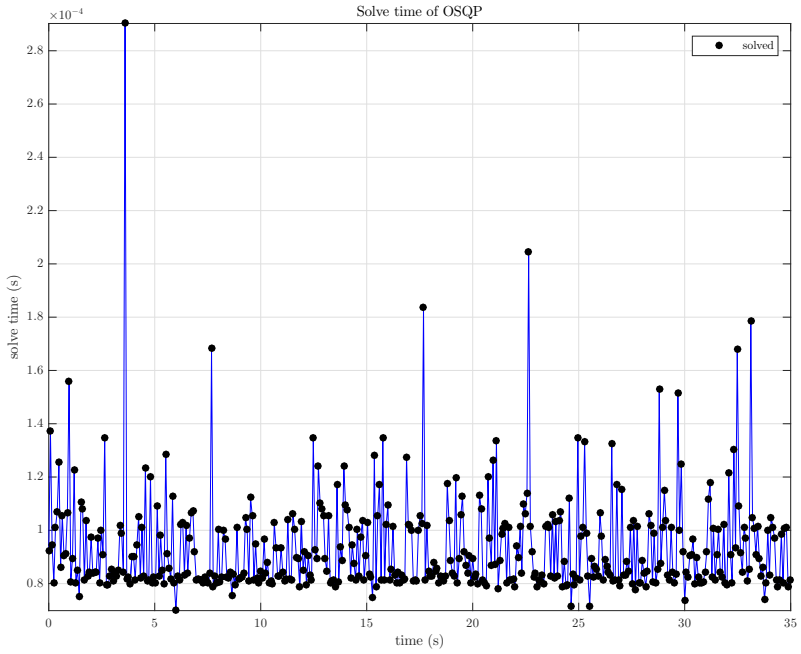

Figure 7.7: OSQP solve time of nominal MPC

A performance of OSQP is illustrated in Figure 7.7. The OSQP has an average solve time of about 0.09 milliseconds, which is about 800 times faster than the frequency at which the MPC has to recompute the control input.

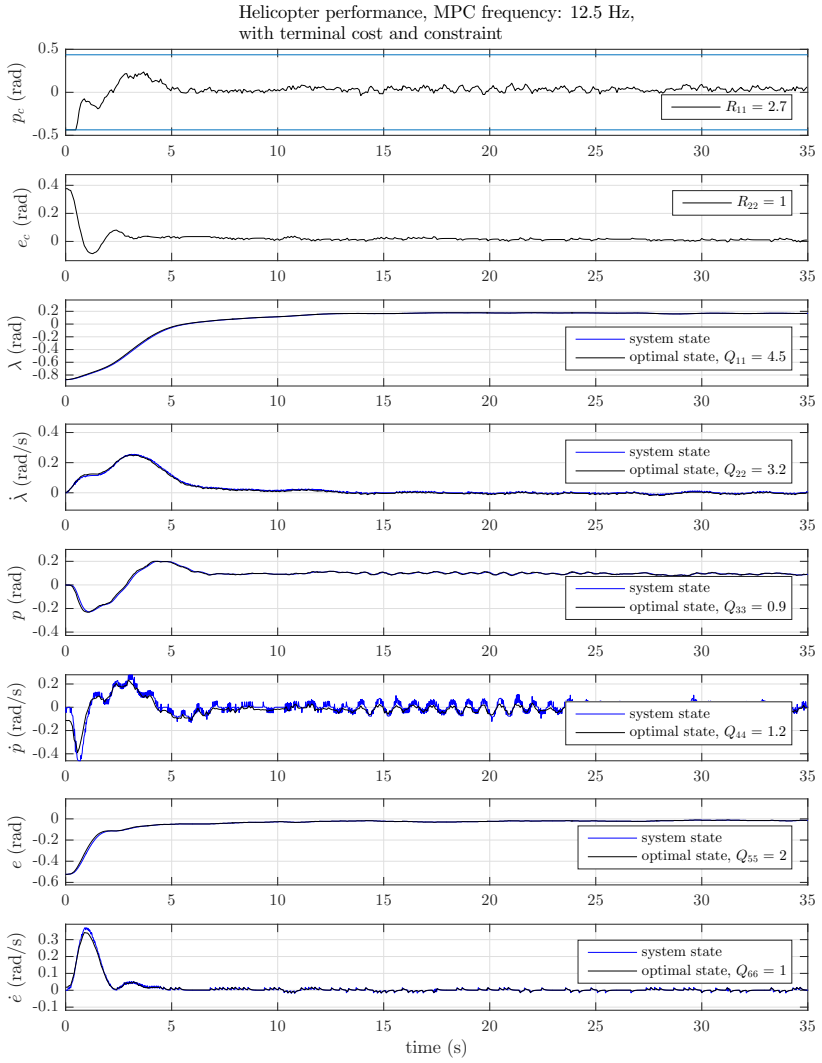

Figure 7.8: Online performance of stable MPC

By adding a terminal cost and constraint, the controller can be tuned much more aggressively without risking an unstable result. Figure 7.8 shows this. Here, λ approaches 0.1675 rad, which is a significant improvement to the nominal performance. The settling time of the system is closer 15 seconds.

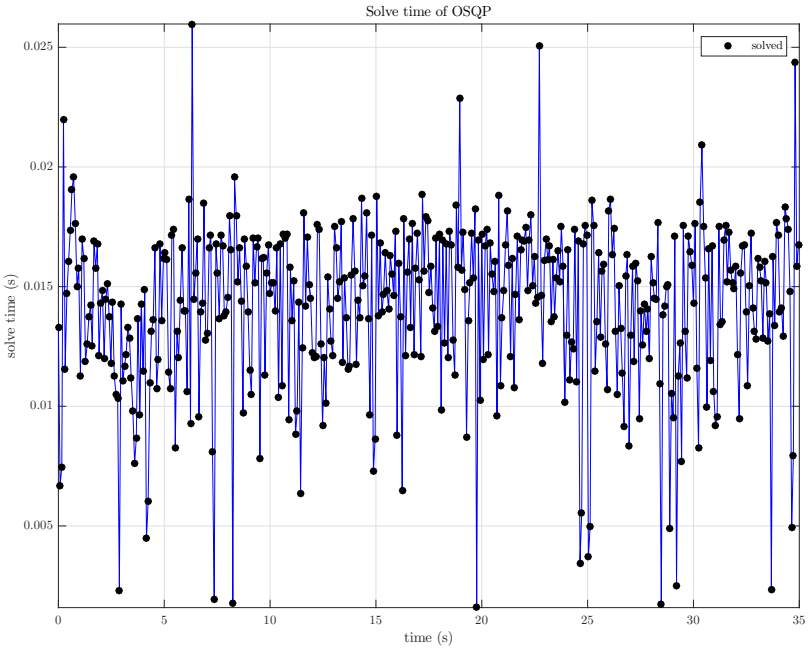

Figure 7.9: OSQP solve time of stable MPC

The solve time of OSQP, shown in Figure 7.9, is now much higher than before. The average solve time is about 14 milliseconds, more than 150 times slower than with nominal MPC. This illustrates the increase of computational complexity when the terminal cost and constraint is added.

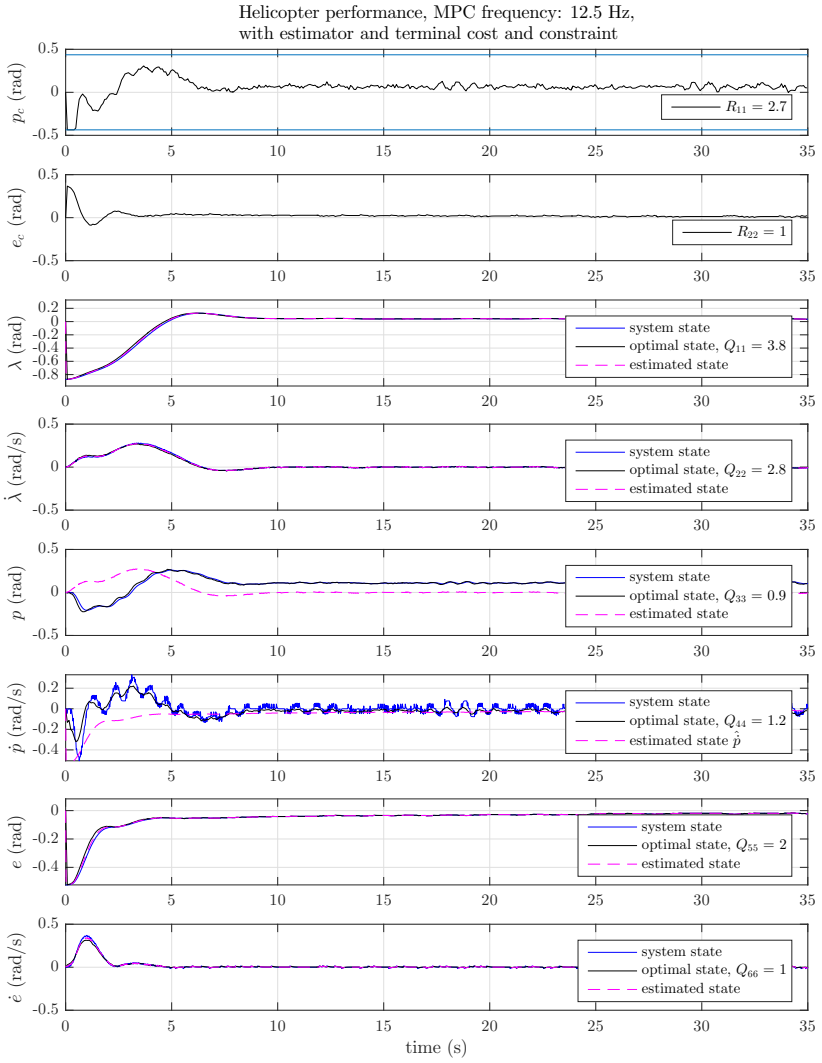

Figure 7.10: Online performance of stable MPC with Kalman filter

Now that the controller is stable, a Kalman filter is added to remove the constant offset. This implementation is successful, seen in Figure 7.10, where λ settles on 0.04 rad. The implementation of a filter also improved the settling time of the system, which is now about 8 seconds.

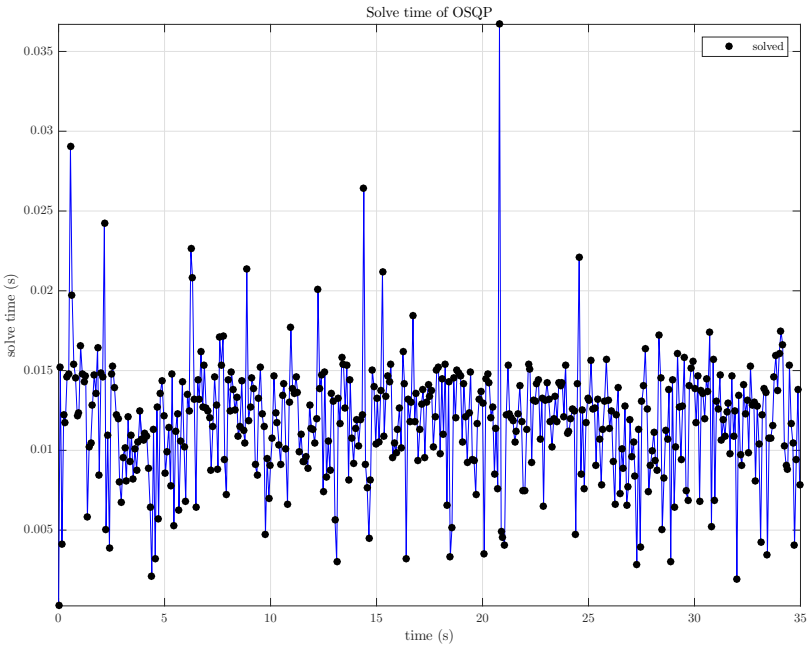

Figure 7.11: OSQP solve time of stable MPC with Kalman filter

The average solve time for this case is 12 milliseconds, as shown in Figure 7.11, which is actually lower than the previous performance without the filter.

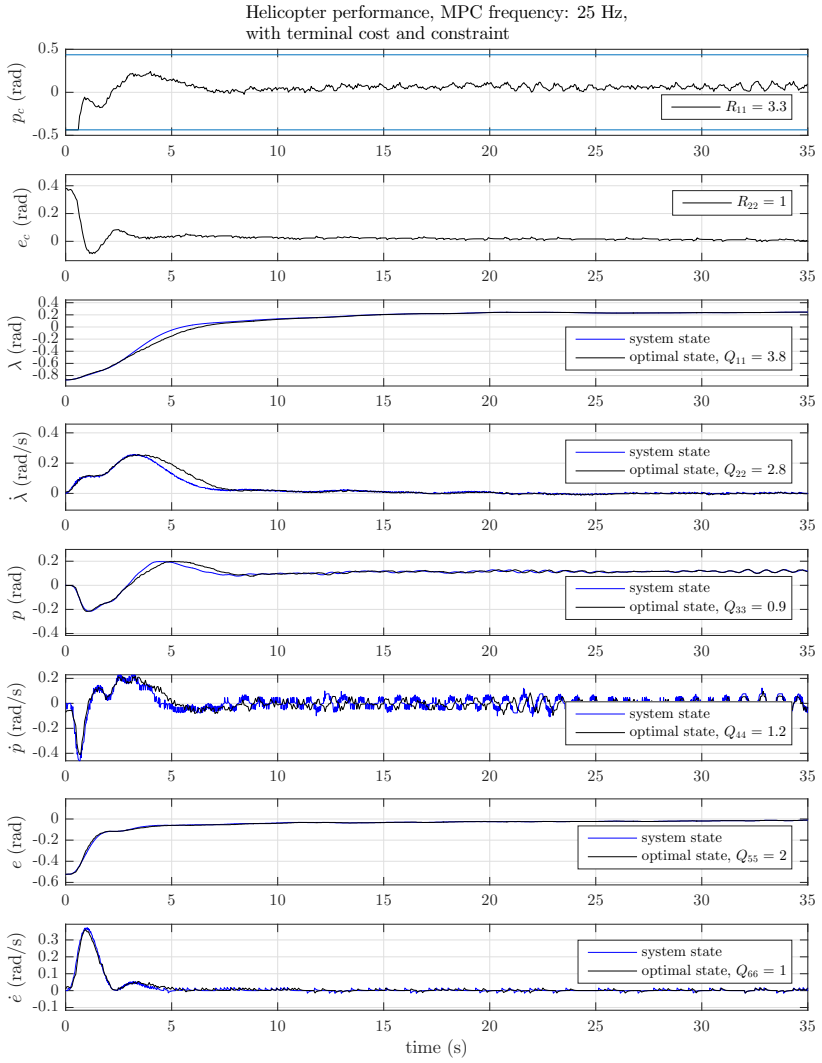

Figure 7.12: Online performance of stable MPC with frequency 25 Hz

The controller was also tested with different frequencies. Figure 7.12 shows the MPC with a sampling time of 0.04 seconds, which is double the sampling time of the previous results. The travel approaches 0.24 rad after about 20 seconds. Since the horizon over which the MPC optimizes should be constant regardless of discretization time, the horizon N has to be doubled, when the frequency is double, so that the controller optimizes over the same time interval.

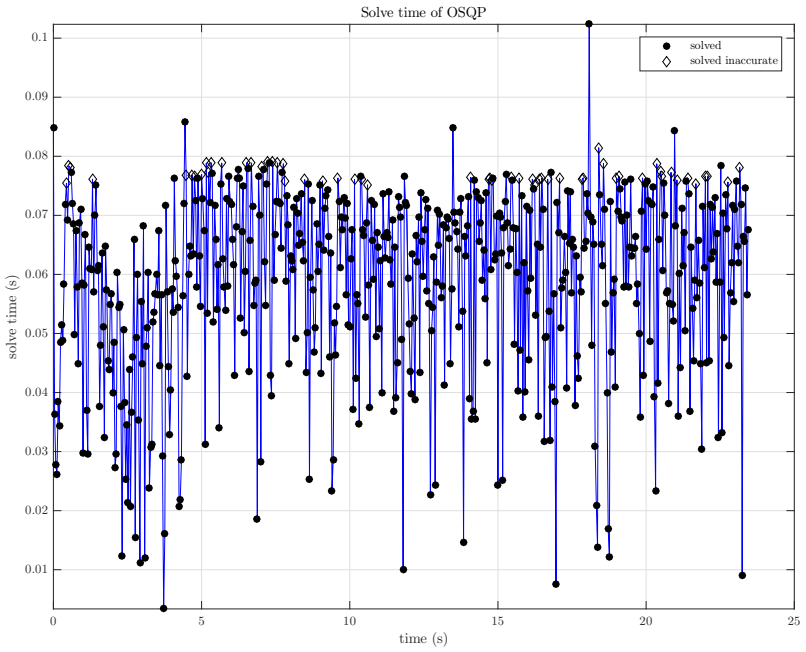

Figure 7.13: OSQP solve time of stable MPC with frequency 25 Hz

Figure 7.13 shows the performance of OSQP during this testing. The average solve time was 59 milliseconds, so a lot of the points are above the sampling time of 0.04 seconds. This means the *actual* frequency at which the MPC returns a control input is lower than 25 Hz. The MPC in this case only returned 586 control input values, while a simulation time of 25 and a sampling interval of 0.04 should return 875.

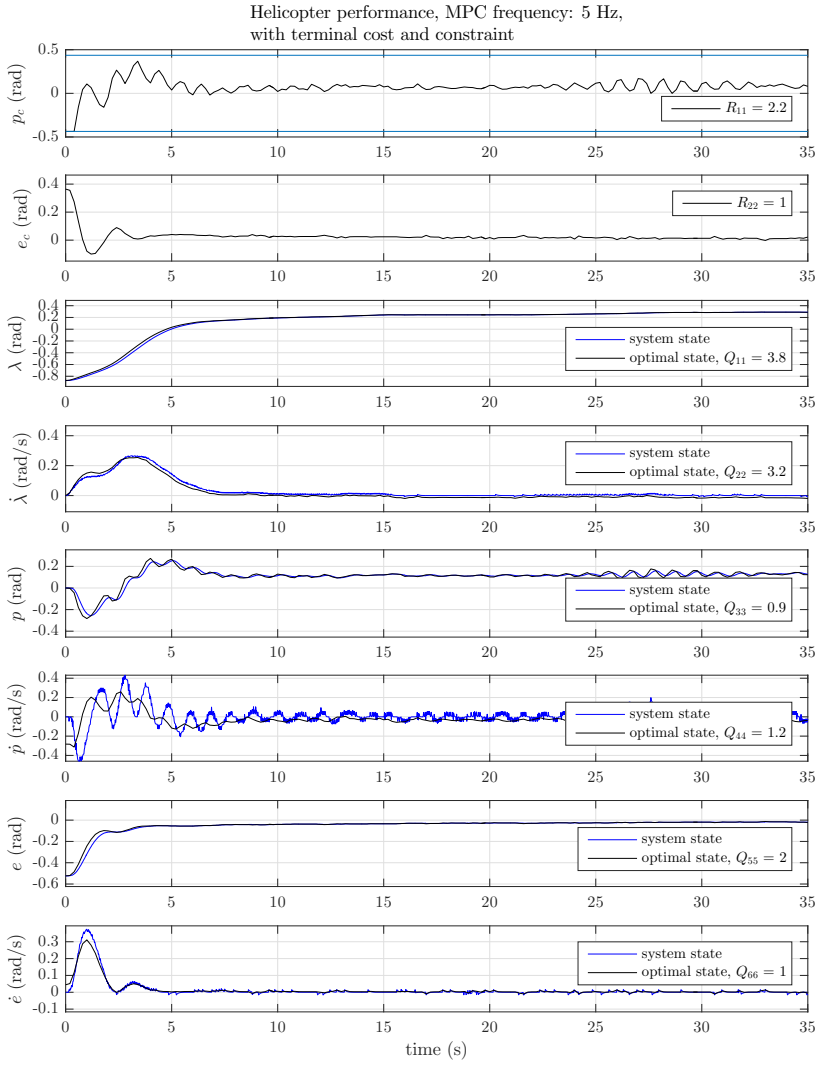

Figure 7.14: Online performance of stable MPC with frequency 5 Hz

The frequency was also lowered to 5 Hz, to see the effects, as shown in Figure 7.14. The travel approaches 0.2889 rad after about 16 seconds. The performance of this controller is not nearly as good as the same controller running at 12.5 Hz. Since lower weighting on the control input resulted in oscillations much easier, the tuning is conservative and the constant deviation is large.

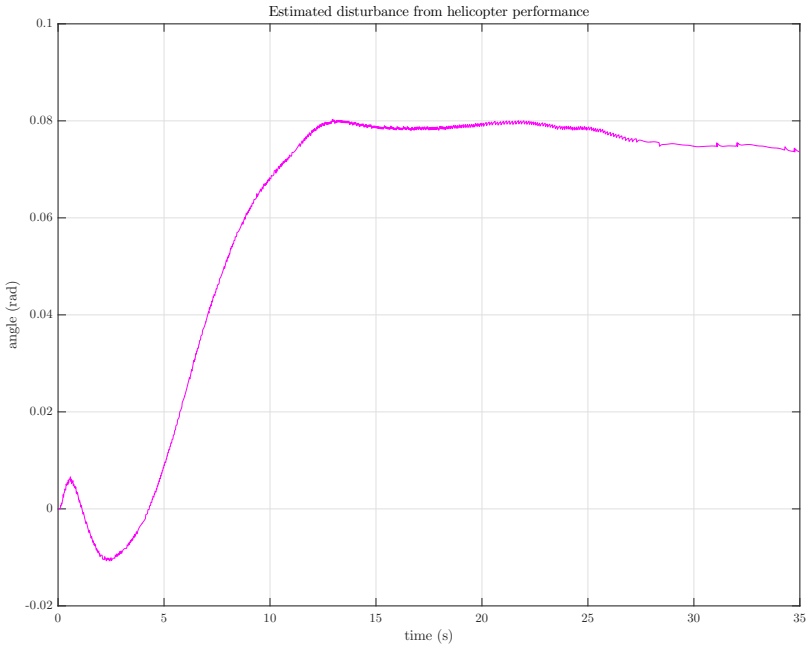

Figure 7.15: Estimated disturbance from Kalman filter

The performance of the Kalman filter can be seen in the helicopter performances with the filter added. However, the most important element of the estimator is the modelled disturbance state d. This estimation is shown in Figure 7.15. The disturbance d settles on 0.075 rad.

Again, a table is added to attempt to quantify the results presented from the numerous experiments on the helicopter. The figures not shown here can be found in Appendix D, specifically the performance of nominal MPC with a filter, and the OSQP solve time of the MPC run at 5 Hz. Since the only weighting variables that vary are the ones related to the travel, travel rate and pitch reference, these are included.

Table 7.2: Results from online performance

controller	settling time	lim λ	avg. solve time	fq	$\begin{bmatrix} Q_{11} \\ Q_{22} \\ R_{11} \end{bmatrix}$
nominal	17 s	0.72 rad	0.09 ms	12.5 Hz	$\begin{bmatrix} 4.5 \\ 3.2 \\ 1.8 \end{bmatrix}$
nominal with filter	17 s	0.54 rad	0.93 ms	12.5 Hz	$\begin{bmatrix} 3.8 \\ 2.8 \\ 0.7 \end{bmatrix}$
stable	15 s	0.17 rad	14.11 ms	12.5 Hz	$\begin{bmatrix} 4.5 \\ 3.2 \\ 2.7 \end{bmatrix}$
stable with filter	8 s	0.04 rad	11.8 ms	12.5 Hz	$\begin{bmatrix} 3.8 \\ 2.8 \\ 2.7 \end{bmatrix}$
stable	20 s	0.24 rad	59.18 ms	25 Hz	$\begin{bmatrix} 3.9 \\ 2.8 \\ 3.3 \end{bmatrix}$
stable	16 s	0.29 rad	2.36 ms	5 Hz	$\begin{bmatrix} 3.8 \\ 3.2 \\ 2.2 \end{bmatrix}$

Chapter 8

Discussion

First and foremost, the MATLAB simulation performs according to the theory presented, which proves the MPC implementation works as intended. Adding the terminal constraint and the terminal cost stabilizes the system, by decreasing both the constant deviation and the settling time (see table 7.1), which was expected, because the MPC now has a hard constraint that steers the state trajectory to a positively invariant set within the horizon. In addition, doubling the optimization horizon N gave the same effect as adding the terminal terms, which was also expected. This is because the terminal cost and constraint are added so the MPC has the same stability properties as the infinite horizon case, without having to increase the horizon, so even though a longer horizon results in the same stability properties, it is computationally costly. This is shown from the average solve time of the solver for both the nominal controller with longer horizon, and the stable controller, shown in table 7.1, where the solve time for the longer horizon is 1.5 times slower. The terminal cost and constraint give the designer the ability to decrease the optimization horizon, resulting in a less complex online control problem.

The implementation of the estimator also resulted in the expected behavior. It was able to model the disturbance generated in the simulation, and remove its effects to a large degree.

The numerical simulation performing as expected makes sense, since the simulation in MATLAB uses the linear helicopter model to calculate the next state of the 'actual system'. Since this same model is used in the

online predictions in the MPC, the system behaves exactly as the controller predicted and control input applied is in fact the optimal input.

When it comes to the performance of the actual helicopter, the results varied. Even though the mere feat of using model predictive control in real time to control the system is a result in itself, the stability of the system can be discussed.

The effect of adding the terminal terms was demonstrated to some degree in real-time, although the settling times did not decrease that much, the travel settles closes to the equilibrium point, from 0.72 to 0.54 radians, see 7.2. Observe that when adding the terminal cost and constraint, the weight of the control input p_c was increased significantly. This is because the existence of the terminal constraint leads to a more aggressive control to ensure the state trajectory ends in the terminal set within the horizon. This type of aggressive control leads to oscillations in both the pitch and pitch reference because of the model inaccuracies. The reason for these oscillations will be discussed in a later paragraph.

Adding the Kalman filter was successful, almost completely removing the constant offset in travel, down to 0.04 radians. However, as can be seen from the estimation of the states, the filter in itself performs rather poorly. This is most likely due to an inaccurate system model being used for the estimator, and admittedly the tuning was not very thorough. However, observe that the estimator without the presence of the terminal terms does little to nothing to remove the constant deviation. This might be due to the fact that the estimator only estimates the assumed disturbance, but it is up to the controller to calculate control inputs that accounts for the disturbance. The presence of the terminal terms creates a much more stable and reliable controller, which will to a large degree be able to account for the estimated disturbance and remove it.

When the Kalman filter was added, the solve time of OSQP decreased slightly from 14 to 12 milliseconds. In theory, the addition of an estimator should not affect the computation time that much, since there is no terminal constraint or cost on the augmented state d. The variation in the solve time is most likely random, and due to the fact that identical controller and parameters can steal lead to different performance of the helicopter since the real world has variations.

The frequency of the MPC was chosen as a trade-off between updating the state of the helicopter in the MPC frequently, and giving the QP-solver enough time to accurately solve the problem. Note the large and small oscillations in the pitch reference p_c, the pitch p and the pitch rate $\dot{p}$. These oscillation are best demonstrated in the pitch rate in Figure 7.7 where the pitch rate in particular almost has standing oscillations. The reason for this is an inaccurate model being used in the MPC calculations. This was shown in Chapter 5 when the nonlinear model was linearized around the equilibrium point. An inaccurate model leads to an inaccurate prediction.

To counteract these oscillations, two solutions are suggested. The first is to increase the weight on the pitch reference, so that the controller does not return such aggressive control inputs. This will limit the oscillations both by limiting the actual value of the control input, and because higher control inputs steer the system away from the equilibrium point, lower input values will avoid doing this, and the model inaccuracy will be reduced. This however, gives a slower system. The other solution to counteracting this is increasing the frequency of the MPC. This will decrease the time between the calculation of the optimal control problem with an updated state, and therefore decrease the time the model inaccuracy has to develop. The solve time of the OSQP determined how much increase is possible.

The selection of the MPC frequency (12.5 Hz) was done with this in mind. A higher frequency would not have been possible because of the run time of OSQP and a lower frequency would mean increasing the weight on the control inputs to avoid oscillations, which gives a slower moving control.

The performance of the QP-solver during the testing is quite impressive. The fastest average was around 0.09 milliseconds, and for the online problem with terminal terms it had an average of 14.1 milliseconds. The solve time of OSQP was the biggest factor when choosing the frequency of the MPC. Here, 12.5 Hz means it is called every 80 milliseconds, which will guarantee a solution before the solver is called again.

When the controller was tested with a higher frequency, 25 Hz, the optimization horizon also had to be doubled. Because to optimize over a constant horizon in time, the optimization horizon N will vary inversely with the sampling frequency. Therefore, lowering the frequency means increasing the optimization horizon in the optimal control problem, thereby increasing the computational cost of the controller. The average solve time was 59

milliseconds, which is higher than the sample time of the OSQP. As a consequence of this, the MPC did not return the expected amount of control inputs, 875, but a significantly lower number. Based on the number of control inputs returner, 586, the simulation time, 35 seconds and the sample time 0.04 seconds, the actual frequency of the MPC in this case was 16.55 Hz.

Also, notice the different solution flags, 'solved' and 'solved inaccurate', in Figure 7.13. The inaccurate solution flag means the algorithm fulfilled the termination criteria, so the solution is within a certain threshold, without being a local minimum. Since OSQP returns high accuracy solutions, returning an inaccurate solution does not mean much for the system performance. Therefore, one could consider increasing the threshold for the termination criteria, without it having a significant impact on the control.

For the controller with a lower frequency, 5 Hz, the oscillations apparent in some of the previous performances was more apparent and easier to induce, by choosing a lower control weight. Due to this, the choice for the weight on the pitch reference is limited. A lower weighting, leads to more aggressive control, and because of the lower frequency, the previously mentioned model inaccuracies have more time to develop before the MPC receives a new state and the control input is updated.

The selection of the optimization horizon N is also a trade-off, between stability and computational time. A larger horizon gives a more stable control, but the online cost and with that the computation time increases. In the case of the stable MPC, the terminal terms emulates the effect a large horizon would have, so the horizon for the online performance was chosen to be $N = 15$. Because of the terminal constraint, the optimization horizon also has to be chosen large enough so that the optimal control input can steer the state trajectory to the terminal set in N steps.

Another aspect worthy of discussion is the computation of the terminal set $\mathbb{X}_f$. The initial implementation of this calculation attempted to compute the MPIS by recalculating the set Ω_i defined in Section 3.2,

$$\Omega_i = \{(x, Kx) \in \Omega_{i-1} \ : \ A_K \cdot (x, Kx) \in (\mathbb{X} \times \mathbb{U})\} \ , \tag{8.1}$$

until it converged. However, after two hours of computing without convergence, this implementation was abandoned. Instead, the implementation used takes advantage of the Multi-Parametric Toolbox 3.0 and its wide range o f functionality to compute the maximal positively invariant set. Because

of the high dimensions in the state space of this particular system, a lot of iterations are necessary before these methods converge, so an optimal implementation of both online and offline calculations is essential.

One of the downsides of this implementation is that for any change in the MPC cost matrices Q and R, the code has to be generated again. The code generation software package wipes the directory clean and generates and links the files again, which takes time. This makes tuning the controller a tedious process.

8.1 Future work

A flaw in the work presented here that is quite obvious, but only revealed once the performance of the helicopter was examined, is the model used in the online optimization. The nonlinear model developed in Chapter 5 was linearized around the equilibrium point. As a consequence, the inaccuracy of the model increases further away from this point, which cause oscillations in some states and limits the frequency selection for the MPC.

Implementing nonlinear MPC would definitely result in a more accurate model and possibly more stable control, but would also involve finding an optimization solver for nonlinear optimization problems, suitable for embedded optimization. A software framework for exactly this purpose has been presented in 2019 by Englert et al. [7], and implemented in both MATLAB and C/C++, with sampling times in the millisecond range.

Adding reference tracking in the MPC is another potential extension that is not out of reach. Using a time-varying signal from Simulink (for example a sine curve) as the reference for the helicopter to follow, would also take advantage of this Simulink functionality.

Another improvement is implementing a form of MPC that accounts for disturbance and is proven to be robust, such as for example tube-based MPC. This will help with the offset in travel, and could also help with the inaccurate model if one can find a way to model the nonlinearities as disturbances.

Put into a larger context, future work involves the continuing development and application of MPC onto similar systems. This implementation could

serve as a basis for other control proposals of MPC or controllers with the same online computational demand. The fact that model predictive control of this helicopter was possible, due to the online calculations being computed fast enough, means further research into this area of MPC has potential and is therefore advantageous investment of time.

Chapter 9

Conclusion

First of all, this thesis is proof-of-concept for linear model predictive control of a helicopter and similar fast moving systems because of the demonstrated performance of the solver. MPC of Quanser's 3 DOF helicopter is made possible by an effective QP solver, OSQP, with support for embedded implementation. The solver performs in the average range of 30 milliseconds per iteration, which is frequent enough for stable control of a helicopter. Additionally, the support for a real-time control software means a software implementation can be created in Simulink, with its pragmatic graphical interface and downloaded onto the embedded hardware.

The addition of a terminal cost and terminal constraint in the open-loop optimal control problem has been shown to have an effect on stability of the closed-loop system, as well as allowing the decrease of the optimization horizon, thereby reducing online complexity. Embedding integral action into the controller by modelling a constant disturbance as an extra state and using that model in the MPC prediction, then adding an estimator to estimate the augmented state, was also demonstrated to be successful.

There are however certain challenges that come with using linear model predictive control, as the system in question is highly nonlinear and the model inaccuracies become clear when the helicopter moves away from the equilibrium point. The introduction of nonlinear model in the prediction could correct this matter, given that the online computation time stays in an appropriately low range.

Using MPC to control a propeller under-actuated system such as the

Quanser's 3 DOF helicopter is both a useful area of research and the proof-of-concept presented in this thesis gives motivation for the development of future implementations of similar controllers, with the same online computational demand as MPC. In conclusion, model predictive control of such a dynamic, fast moving system means the application of MPC can continue on to similar systems and has also demonstrated the potential impact this technology can have on industries that rely on this type of control.

Acronyms

ADMM	alternating direction method of multipliers.
DARE	discrete-time algebraic Riccati equation.
DOF	degrees of freedom.
EMPC	explicit model predictive control.
HIL	Hardware-in-the-loop.
IO	input-output.
KKT	Karush–Kuhn–Tucker.
LQE	linear-quadratic estimator.
LQR	linear-quadratic regulator.
MIMO	multi-input multi-output.
MPC	model predictive control.
MPIS	maximal positively invariant set.
NMPC	nonlinear model predictive control.
OSQP	Operator Splitting Quadratic Program.
PID	proportional–integral–derivative.
QP	quadratic programming.

Bibliography

[1] Arican, A. C., Ozcan, S., Kocagil, B. M., Guzey, U. M., Copur, E. H., Salamci, M. U., 2018. Linear and nonlinear optimal controller design for a 3 DOF helicopter. In: 2018 19th International Carpathian Control Conference (ICCC). IEEE, pp. 185–190.

[2] Banjac, G., Stellato, B., Moehle, N., Goulart, P., Bemporad, A., Boyd, S., Dec. 2017. Embedded code generation using the OSQP solver. In: IEEE Conference on Decision and Control (CDC).
URL `https://doi.org/10.1109/CDC.2017.8263928`

[3] Bemporad, Alberto, P. P., 08 2012. Simple and certifiable quadratic programming algorithms for embedded linear model predictive control. Vol. 4. pp. 14–20.

[4] Binder, B. J. T., Kufoalor, D. K. M., Johansen, T. A., Sep. 2015. Scalability of qp solvers for embedded model predictive control applied to a subsea petroleum production system. In: 2015 IEEE Conference on Control Applications (CCA). pp. 1173–1178.

[5] Boyd, S., Parikh, N., Chu, E., Peleato, B., Eckstein, J., 01 2011. Distributed optimization and statistical learning via the alternating direction method of multipliers. Foundations and Trends in Machine Learning 3, 1–122.

[6] de Nicolao, G., Magni, L., Scattolini, R., March 1996. On the robustness of receding-horizon control with terminal constraints. IEEE Transactions on Automatic Control 41 (3), 451–453.

[7] Englert, T., Völz, A., Mesmer, F., Rhein, S., Graichen, K., Jan 2019. A software framework for embedded nonlinear model predictive control using a gradient-based augmented lagrangian approach (GRAMPC).

Optimization and Engineering.
URL https://doi.org/10.1007/s11081-018-9417-2

[8] Ferreau, H. J., Almer, S., Verschueren, R., Diehl, M., Frick, D., Domahidi, A., Jerez, J. L., Stathopoulos, G., Jones, C., july 2017. Embedded optimization methods for industrial automatic control. 20th IFAC World Congress 50, 13194–13209.

[9] Foss, B., Heirung, T., 2016. Merging optimization and control.

[10] Ghadimi, E., Teixeira, A., Shames, I., Johansson, M., March 2015. Optimal parameter selection for the alternating direction method of multipliers (admm): Quadratic problems. IEEE Transactions on Automatic Control 60 (3), 644–658.

[11] Herceg, M., Kvasnica, M., Jones, C., Morari, M., July 17–19 2013. In: Proc. of the European Control Conference. Zürich, Switzerland, pp. 502–510, http://control.ee.ethz.ch/~mpt.

[12] Ju, Z., Xinyan, C., 2014. Explicit model predictive control of 3-DOF helicopter, 3083 – 3088.

[13] Kerrigan, E. C., Khusainov, B., Constantinides, G. A., June 2016. What is different about embedded optimization? In: 2016 European Control Conference (ECC). pp. 600–600.

[14] Kocagil, B. M., Ancan, A. C., Guzey, U. M., Ozcan, S., Salamci, M. U., 2017. Controller designs for nonlinear systems with application to 3 DOF helicopter model. Gazi University Journal of Science, Part A: Engineering and Innovation 4 (3), 47–66.

[15] Maciejowski, J. M., 2001. Predictive Control with Constraints. Prentice Hall, Harlow.

[16] MathWorks, 2019. S-function.
URL https://www.mathworks.com/help/simulink/slref/sfunctionbuilder.html

[17] Propoi, A. I., 1963. Application of linear programming methods for the synthesis of automatic sampled-data systems. Avtomat. i Telemekh 24, 912–920.

[18] Qin, J., Badgwell, T., 07 2003. A survey of industrial model predictive control technology. Control engineering practice 11, 733–764.

[19] Quanser, 2019. 3 DOF Helicopter. https://www.quanser.com/products/3-dof-helicopter/, accessed: 2019-06-01.

[20] Quanser, 2019. Multithreading.
URL http://quanser-update.azurewebsites.net/quarc/documentation/quarc_multithreading.html

[21] Rawlings, J., Mayne, D., 2009. Model Predictive Control: Theory, Computation, and Design. Nob Hill Publishing.

[22] Rawlings, J., Mayne, D., Rao, C., Scokaert, P., 2000. Constrained model predictive control: Stability and optimality. Automatica 36, 789–814.

[23] Richalet, J., Abu El Ata-Doss, S., Arber, C., Kuntze, H., Jacubasch, A., Schill, W., 07 1987. Predictive functional control - application to fast and accurate robots. IFAC Proceedings Volumes 20, 251–258.

[24] Richalet, J., Rault, A., Testud, J. L., Papon, J., 1976. Algorithmic control of industrial processes 10, 1119–1167.

[25] Richalet, J., Rault, A., Testud, J. L., Papon, J., 1978. Model predictive heuristic control: Applications to industrial processes. Automatica 14, 413–428.

[26] Sollie, H.-I. H., 2018. Tube-based methods for robust linear MPC. NTNU Project report.

[27] Stellato, B., Banjac, G., Goulart, P., Bemporad, A., Boyd, S., Nov. 2017. OSQP: An operator splitting solver for quadratic programs. ArXiv e-prints.

[28] Sznaier, M., Damborg, M. J., Dec 1987. Suboptimal control of linear systems with state and control inequality constraints. In: 26th IEEE Conference on Decision and Control. Vol. 26. pp. 761–762.

[29] Veeraboina, A.K.; Ordonez, R., 2018. Design and implementation of linear/nonlinear control methods on 3-DOF helicopter. In: IEEE National Aerospace and Electronics Conference (NAECON) 2018. pp. 435–442.

[30] Yang, H., Jiang, B., Liu, H. H. T., Yang, H., Zhang, Q., 2019. Attitude

synchronization for multiple 3-DOF helicopters with actuator faults. IEEE/ASME Transactions on Mechatronics.

[31] Zhai, Y., 2013. Robust observer based model predictive control of a 3-DOF helicopter system. In: Jung, H.-K., Kim, J. T., Sahama, T., Yang, C.-H. (Eds.), Future Information Communication Technology and Applications. Vol. 235 of Lecture Notes in Electrical Engineering. Springer Netherlands, Ch. 11, pp. 89–100.

[32] Zhai, Y., Nounou, M., Nounou, H., Al-Hamidi, Y., 2010. Model predictive control of a 3-DOF helicopter system using successive linearization. International Journal of Engineering, Science and Technology 2 (10), 9–19.

[33] Zhang, J., Cheng, X., Zhu, J., 2016. Control of a laboratory 3-DOF helicopter: Explicit model predictive approach. International Journal of Control, Automation and Systems 14 (2), 389–399.

Appendices

Appendix A

System parameters

Table A.1: Helicopter Parameters and Values

Symbol	Parameter	Value	Unit
g	Gravitational acceleration	9.81	m / s
l_c	Distance from elevation axis to helicopter body	0.65	m
l_p	Distance from pitch axis to motor	0.17	m
K_f	Motor force constant	0.0446	N / V
J_e	Moment of inertia for elevation	0.338	kg m^2
J_λ	Moment of inertia for travel	0.338	kg m^2
J_p	Moment of inertia for pitch	0.0116	kg m^2
m_p	Mass of one motor	0.2	kg
m_g	Effective mass of the helicopter	0.03	kg
V_s^*	Voltage sum equilibrium	6.4	V
V_d^*	Voltage difference equilibrium	0.35	V

Table A.2: Controller Parameters and Values

Symbol	Parameter	Value
w_p	Bandwidth for pitch controller	1.8
d_p	Relative damping for pitch controller	1.0
w_e	Bandwidth for elevation controller	0.5
d_e	Relative damping for elevation controller	1.0
K_{pp}	Proportional constant for pitch controller	4.7163
K_{pd}	Derivative constant for pitch controller	5.2404
K_{ep}	Proportional constant for elevation controller	2.7829
K_{ed}	Derivative constant for elevation controller	11.1315
K_{ei}	Integral constant for elevation controller	0.2783

Appendix B

MATLAB Code

Listing B.1: execute.m

```
1
2  clear x u v z x_est x_opt
3
4  %% parameters
5  system.Nsim       = 650;  % simulation time
6  system.stable     = 0;    % enable terminal cost and terminal
       constraint
7  with_estimator    = 1;    % enable Kalman filter
8  run_sim           = 0;    % enable simulation
9  run_heli          = 1;    % enable code generation
10
11 %% load system data
12 init;
13 data;
14
15 %% MPC objective function
16 % weighting
17 % system states
18 q_travel          = 3.8;   % for travel
19 q_travel_rate     = 2.8;   % for travel rate
20 q_pitch           = 0.9;   % for pitch
21 q_pitch_rate      = 1.2;   % for pitch rate
22 q_elevation       = 2;     % for elevation
23 q_elevation_rate  = 1;     % for elevation rate
24 q_dist            = 0;     % for modelled disturbance
25 % control inputs
```

```
26 r_pitch_ref       = 1.2;   % for pitch ref
27 r_elevation_ref   = 1;     % for elevation ref
28 cost.S            = 0.3;   % for slack variable
29
30 %generate matrices Q, Q_heli, R
31 cost.Q = diag([q_travel     q_travel_rate     ...
32                q_pitch      q_pitch_rate      ...
33                q_elevation  q_elevation_rate  ...
34                q_dist]);
35 cost.R = diag([r_pitch_ref r_elevation_ref]);
36
37 %% kalman filter
38 % create matrices Q_K, R_K
39 kalman_filter;
40
41 %% offline calculations
42 % create matrices K, P, X_f, Su, Sx
43 offline_calc;
44
45 %% assign to problem
46 problem.system = system;
47 problem.constraints = constraints;
48 problem.cost = cost;
49 problem.estimator = estimator;
50 problem.disturbance = disturbance;
51 [problem.mpc_cost, problem.mpc_constraints] =
       gen_mpc_matrices(problem); % generate matrices for mpc
52
53 %% assign to problem
54 problem.system                              = system;
55 problem.constraints                         = constraints;
56 problem.cost                                = cost;
57 problem.estimator                           = estimator;
58 problem.disturbance                         = disturbance;
59 [problem.mpc_cost, problem.mpc_constraints] =
       gen_mpc_matrices(problem); % generate matrices for mpc
60
61 %% initial value
62 travel_offset     = -50;   % for travel
63 elevation_offset  = -30;   % for elevation
64 L_model           = L_K;   % assign gain to estimator in
       model
65
```

```
66  if (run_heli)
67  %% generate code for S-Function
68  % make sure all files in the 'c_code'-directory are closed
69  gen_code_osqp;          % generate code based on problem data
70  gen_code_params;        % generate strings for bin, cin, f,
        Su, Sx,
71                          % NUM_CONSTRAINTS, LENGTH_Z, LENGTH_X
                                , LENGTH_U for 'workspace.h'
72
73  end
74
75  if (run_sim)
76  %% disturbance
77  w_sequence = gen_disturbance(problem);
78
79  %% initial value
80  travel_offset      = -50; % for travel
81  elevation_offset   = -30; % for elevation
82  L_model            = L_K;    % assign gain to estimator in
        model
83  system.x0          = [    travel_offset * DEGTORAD;  0;
84                                                  0;  0;
85                        elevation_offset * DEGTORAD;  0;
86                                                     0];
87
88  %% simulation
89  for i = 1 : system.Nsim
90      % with kalman filter
91      if (with_estimator)
92          optimal(i)        = online_calc(problem,x_est(:,i));
93          u(:,i)            = optimal(i).u(:,1);
94          x_opt(:,i)        = optimal(i).x(:,1);
95          e_flag(i)         = optimal(i).e;
96          run_time_sim(i)   = optimal(i).run_time;
97          x_est(:,i+1)      = problem.system.A * x_est(:,i) +
                problem.system.B * u(:,i) ...
98                            + L_K * system.C * (x(:,i) - x_est
                                  (:,i));
99          if ~(i == system.Nsim)
100                 x(:,i+1) = problem.system.A * x(:,i) +
                        problem.system.B * u(:,i) ...
101                           + system.h * E * w_sequence(:,i);
102         end
```

```
103     % without kalman filter
104     else
105         optimal(i)        = online_calc(problem,x(:,i));
106         u(:,i)            = optimal(i).u(:,1);
107         x_opt(:,i)        = optimal(i).x(:,1);
108         e_flag(i)         = optimal(i).e;
109         run_time_sim(i)   = optimal(i).run_time;
110         if ~(i == system.Nsim)
111                 x(:,i+1) = problem.system.A * x(:,i) +
                        problem.system.B * u(:,i) ...
112                         + system.h * E * w_sequence(:,i);
113         end
114     end
115 end
116 end
```

Listing B.2: offline_calc.m

```
1  % state feedback and terminal cost
2  [K, P]  = dlqr(system.A_heli, ...
3                 system.B_heli, ...
4                 cost.Q(1:system.n-1, 1:system.n-1), ...
5                 cost.R);
6
7  % extend to add system state d
8  system.K = [-K zeros(system.m, 1)];
9  cost.P = [P zeros(size(P,1), 1) ;
10           zeros(1, size(P,1)+1)];
11 cost.P(system.n, system.n) = 0;
12
13 % generate matrices Sx, Su
14 [Sx, Su] = genMPCprob(system.A,system.B,system.N);
15 system.Sx = Sx;
16 system.Su = Su;
17
18 % calculate terminal constraint set
19 % if already exists
20 if exist([pwd '\terminal_constraint.mat'], 'file') == 2
21         load 'terminal_constraint.mat'
22         constraints.G = X_f.A;
23         constraints.h = X_f.b;
24 % if not, calculate
25 else
```

```
26          %define matrices for system without augmented state
27          constraints.C_heli = [1  0  0  0  0  0;
28                                -1  0  0  0  0  0;
29                                 0  0  1  0  0  0;
30                                 0  0 -1  0  0  0;
31                                zeros(system.m, system.n-1)];
32          constraints.D_heli = [ zeros(size(constraints.C,1) -
                system.m, system.m) ;
33                                1  0;
34                                -1  0];
35          constraints.e_heli = DEGTORAD * [c_travel;
                c_travel;
36                                           c_pitch;
                                                c_pitch;
37                                           c_pitch_ref;
                                                c_pitch_ref];
38
39          constraints.C_K_heli        = constraints.C_heli -
                constraints.D_heli * K;
40          system.A_K_heli             = system.A_heli      -
                system.B_heli       * K;
41          constraints.P               = Polyhedron('A',
                constraints.C_K_heli, 'b', constraints.e_heli);
42          model_heli                  = LTISystem('A',system.
                A_K_heli);
43          model_heli.x.with('setConstraint');
44          model_heli.x.setConstraint = constraints.P;
45          X_f_heli                    = model_heli.invariantSet
                ();
46          % no terminal constraint on d, just zeros
47          constraints.G = [X_f_heli.A, zeros(size(X_f_heli.A,1)
                , 1)];
48          constraints.h = X_f_heli.b;
49          X_f = Polyhedron(constraints.G, constraints.h);
50          save('terminal_constraint', 'X_f')
51  end
```

Listing B.3: kalman_filter.m

```
% process noise
w_travel          = 0.35;   % for travel
w_travel_rate     = 0.1;    % for travel rate
w_pitch           = 0.058;  % for pitch
w_pitch_rate      = 0.41;   % for pitch rate
w_elevation       = 0.2;    % for elevation
w_elevation_rate  = 0.1;    % for elevation rate
w_dist            = 0.15;   % for modelled disturbance
% measurement noise
v_travel          = 0.02;   % for travel
v_travel_rate     = 0.01;   % for travel rate
v_pitch           = 0.06;   % for pitch
v_pitch_rate      = 0.07;   % for pitch rate
v_elevation       = 0.012;  % for elevation
v_elevation_rate  = 0.01;   % for elevation rate

% generate matrices Q_K, R_K
estimator.Q_K = diag([w_travel     w_travel_rate     ...
                      w_pitch      w_pitch_rate      ...
                      w_elevation  w_elevation_rate  ...
                                   w_dist]);

% measurement noise covariance matrix
estimator.R_K = diag([v_travel     v_travel_rate  ...
                      v_pitch      v_pitch_rate   ...
                      v_elevation  v_elevation_rate]);

% calculate Kalman filter gain
plant = ss(system.A, [system.B system.G], system.C, [system.
    D system.H], -1);
[~, L_K, ~] = kalman(plant, estimator.Q_K, estimator.R_K);
```

Listing B.4: gen_mpc_matrices.m

```
function [mpc_cost, mpc_constraints] = generate_mpc_matrices
    (problem)

N = problem.system.N;
n = problem.system.n;
m = problem.system.m;
Sx = problem.system.Sx;
Su = problem.system.Su;
```

```
8
9  if (problem.system.stable)
10     C_tilde = [kron(eye(N), problem.constraints.C);
11                zeros(size(problem.constraints.G,1), n * (N
                       -1)) problem.constraints.G];
12     D_tilde = [kron(eye(N), problem.constraints.D); zeros(
           size(problem.constraints.G,1), m*N)];
13     E_tilde = [kron(eye(N), problem.constraints.E); zeros(
           size(problem.constraints.G,1), N)];
14     e_tilde = [kron(ones(N,1),problem.constraints.e);
           problem.constraints.h];
15     Q_big   = blkdiag(kron(eye(N-1), problem.cost.Q),
           problem.cost.P);
16 else
17     C_tilde = kron(eye(N), problem.constraints.C);
18     D_tilde = kron(eye(N), problem.constraints.D);
19     E_tilde = kron(eye(N), problem.constraints.E);
20     e_tilde = kron(ones(N,1),problem.constraints.e);
21     Q_big   = blkdiag(kron(eye(N), problem.cost.Q));
22 end
23
24 R_big = kron(eye(problem.system.N), problem.cost.R);
25 Q_R_tilde = Su' * Q_big * Su + R_big;
26 S_tilde = kron(eye(N), problem.cost.S);
27 H = [Q_R_tilde zeros(size(Q_R_tilde,1), size(S_tilde,2)) ;
28       zeros(size(S_tilde,1), size(Q_R_tilde,1)) S_tilde ];
29 f = [2 * Sx' * Q_big * Su zeros(n, N)];
30
31 mpc_constraints.Ain = [C_tilde * Su + D_tilde E_tilde];
32 mpc_constraints.bin = e_tilde;
33 mpc_constraints.cin = -C_tilde * Sx;
34 mpc_cost.H = 2 * H;
35 mpc_cost.f = f;
```

Listing B.5: gen_code_osqp.m

```
1
2  n     = problem.system.n;
3  m     = problem.system.m;
4  N     = problem.system.N;
5
6  %irrelevant value, will be updated
7  x0_osqp = [1; 1; 1; 1; 1; 1; 1];
8
9  P     = problem.mpc_cost.H;
10 q     = (x0_osqp' * problem.mpc_cost.f)';
11 A     = [problem.mpc_constraints.Ain];
12 lb    = -Inf * ones(size(problem.mpc_constraints.Ain,1), 1);
13 ub    = problem.mpc_constraints.bin + problem.mpc_constraints
       .cin * x0_osqp;
14
15 %% set up osqp problem
16 qp_problem         = osqp;                                % create
       osqp object
17 settings          = qp_problem.default_settings();  % obtain
       defualt settings
18 settings.verbose  = 0;                                   %
      disable printing output
19 qp_problem.setup(   H, ...
20                     q, ...
21                     A, lb, ub, ...
22                     settings);
23
24  %% generate code
25  % directory name - if changing this name, remember to
       change the name in 'S-Function Builder'
26 dir_name = 'c_code';
27
28 qp_problem.codegen(dir_name,                              ...
29                    'project_type',    'Makefile',     ...
30                    'parameters',      'matrices',     ...
31                    'force_rewrite',   true);
32
33 %% move files that are generated
34 current_path = pwd;
35 file_names = {'auxil.h',            'constants.h',    '
      glob_opts.h',   ...
36                 'kkt.h',             'lin_alg.h',      'osqp.h'
```

```
                ,       ...
37              'osqp_configure.h',  'proj.h',        'qdldl.h
                ',       ...
38              'qdldl_interface.h', 'qdldl_types.h', 'scaling
                .h',     ...
39              'types.h',           'util.h',        '
                workspace.h' };
40
41  for i = 1:size(file_names,2)
42      copyfile( [current_path '\' dir_name '\include\' char(
            file_names(i))], ...
43                [current_path '\' char(file_names(i))]) ;
44      copyfile( [current_path '\' dir_name '\include\' char(
            file_names(i))], ...
45                [current_path '\' dir_name '\src\osqp\' char(
                    file_names(i))]) ;
46  end
```

Appendix C

Simulink model

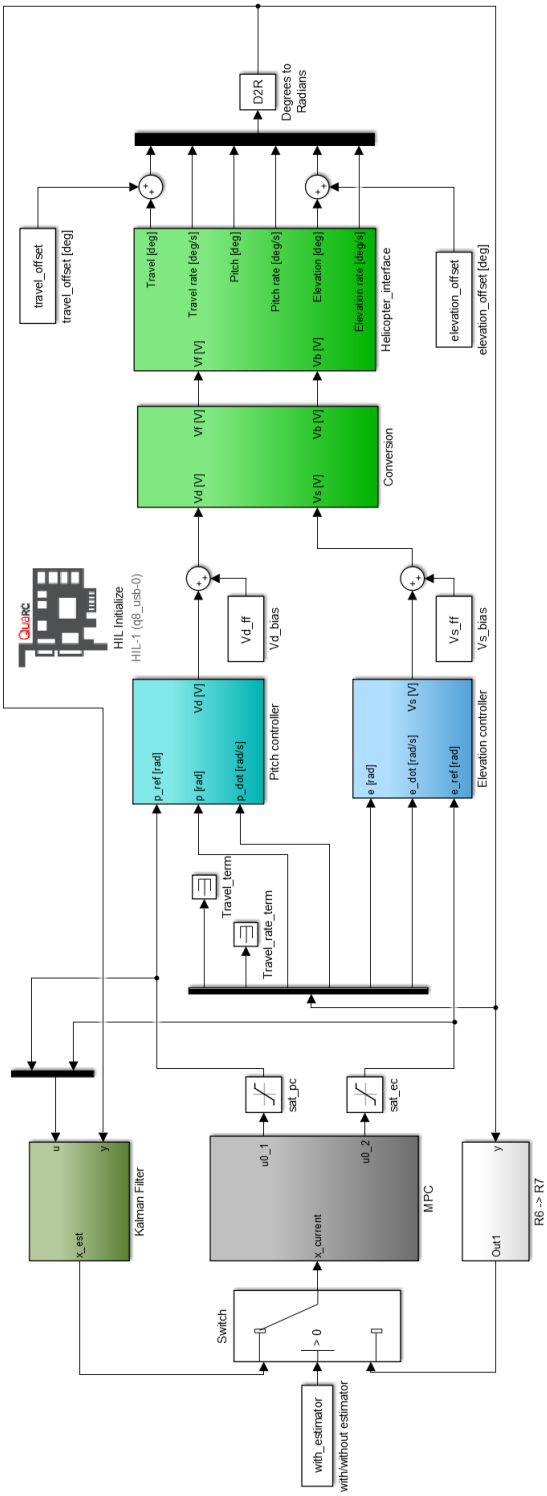

Figure C.1: Simulink model

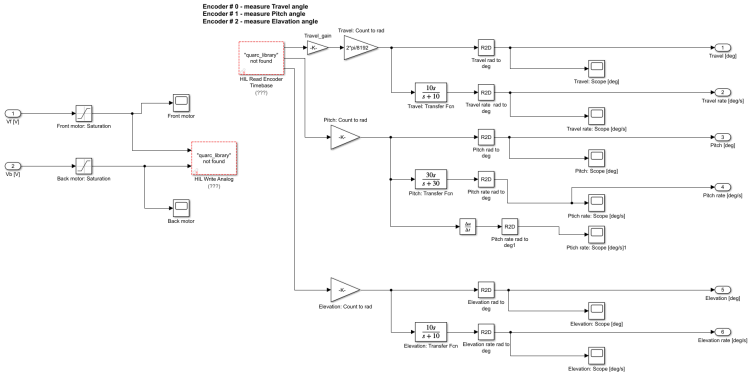

Figure C.2: Helicopter interface

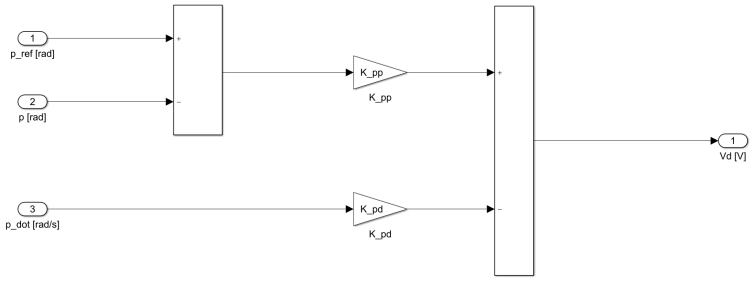

Figure C.3: Pitch controller

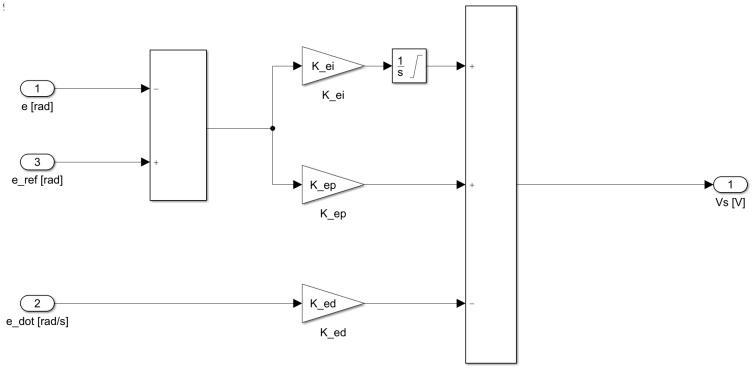

Figure C.4: Elevation controller

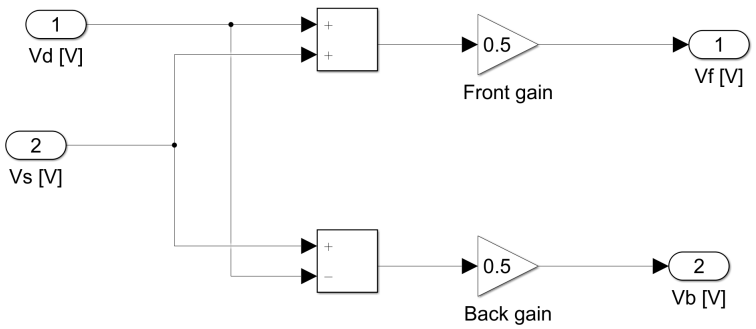

Figure C.5: Voltage conversion block

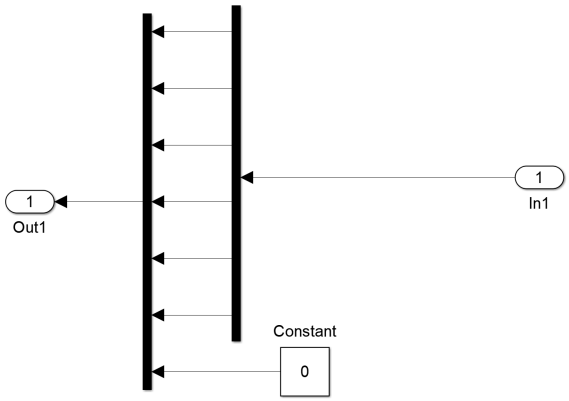

Figure C.6: R6 -> R7 block

Appendix D

Additional figures from results

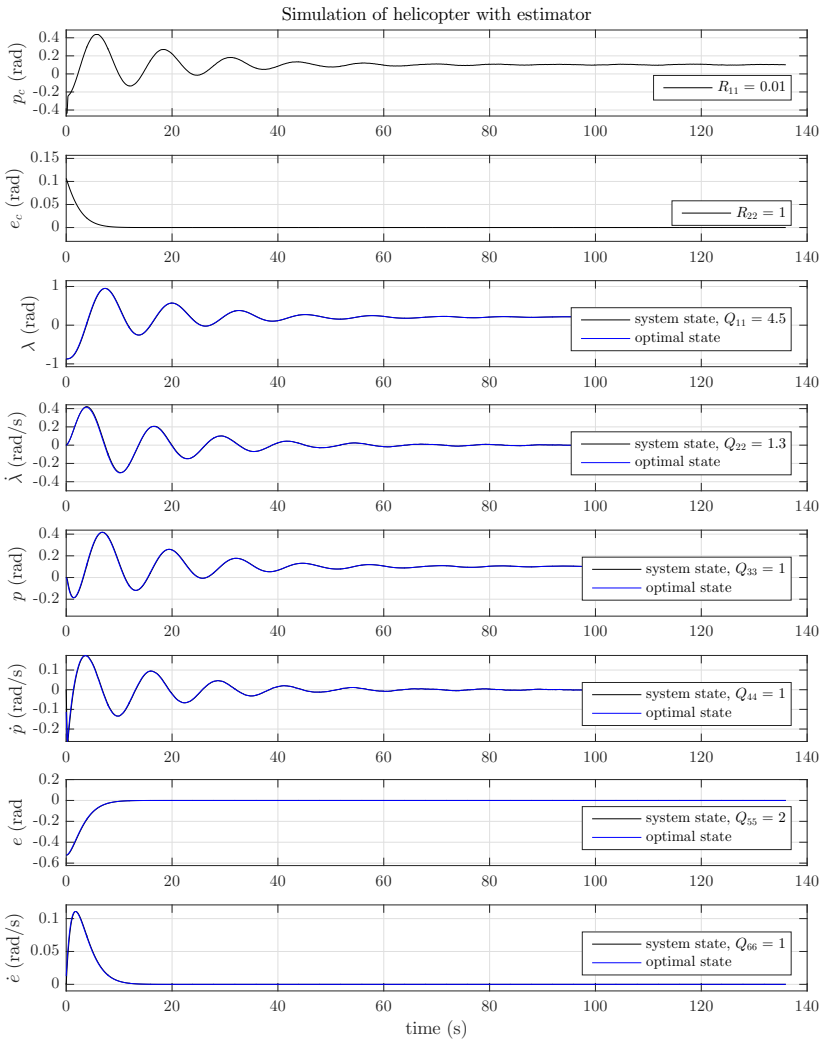

Figure D.1: Simulation of nominal MPC with Kalman filter

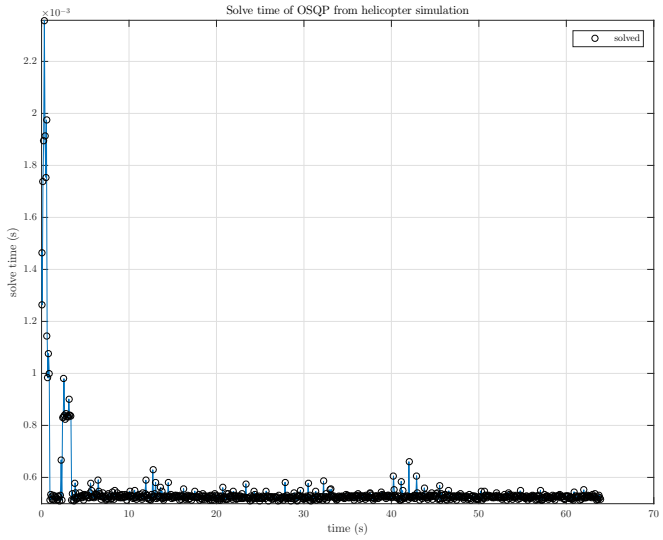

Figure D.2: OSQP solve time of simulation of nominalMPC with Kalman filter

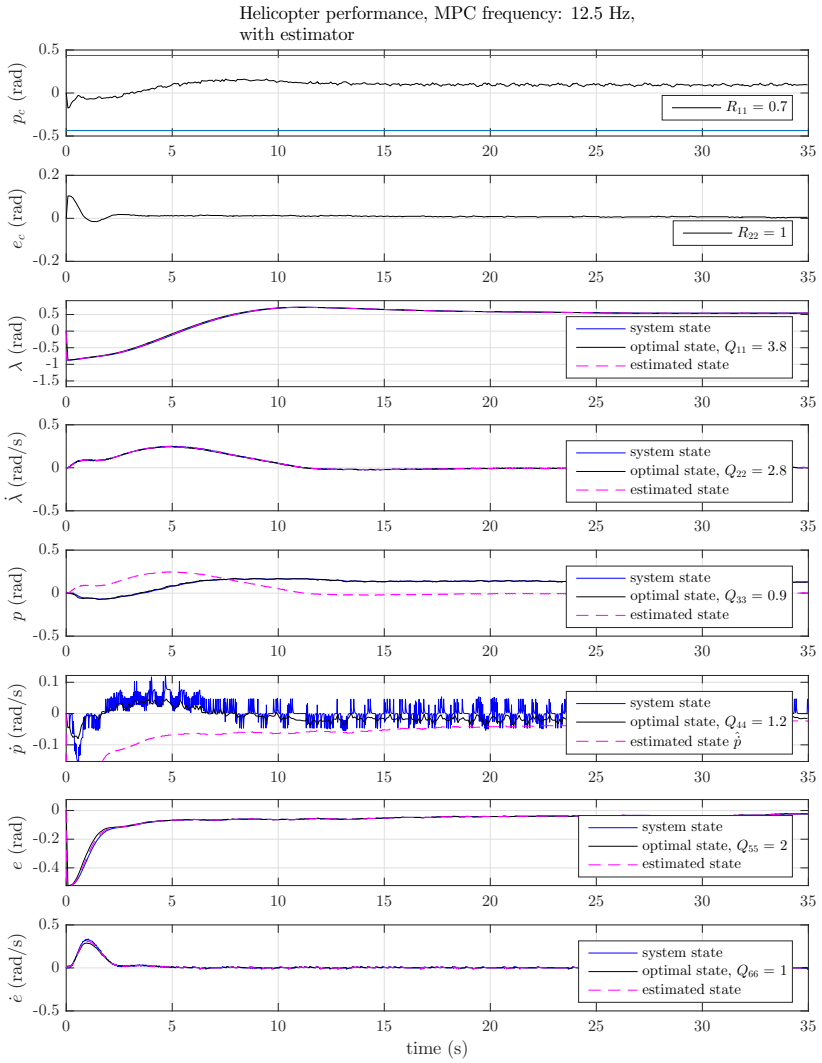

Figure D.3: Online performance of nominal MPC with Kalman filter

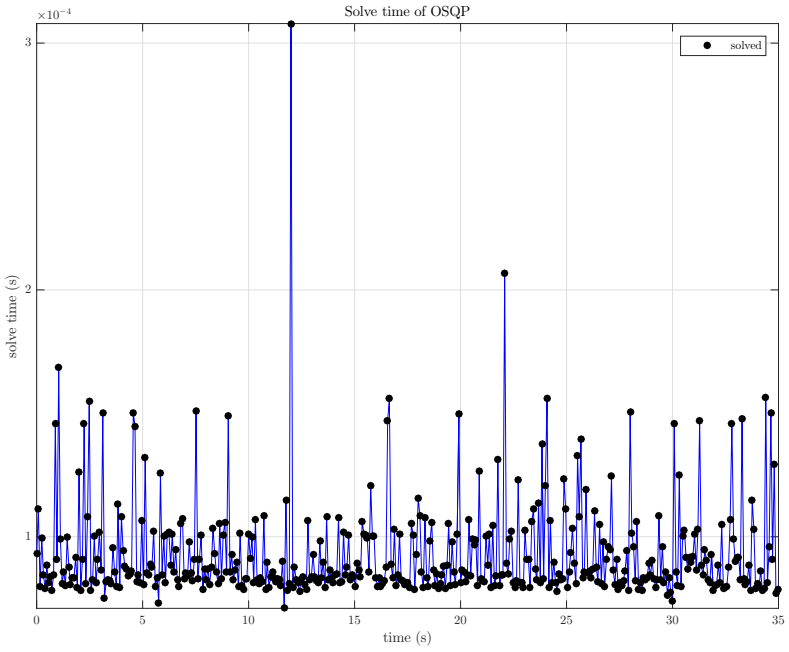

Figure D.4: OSQP solve time of nominal MPC and Kalman filter

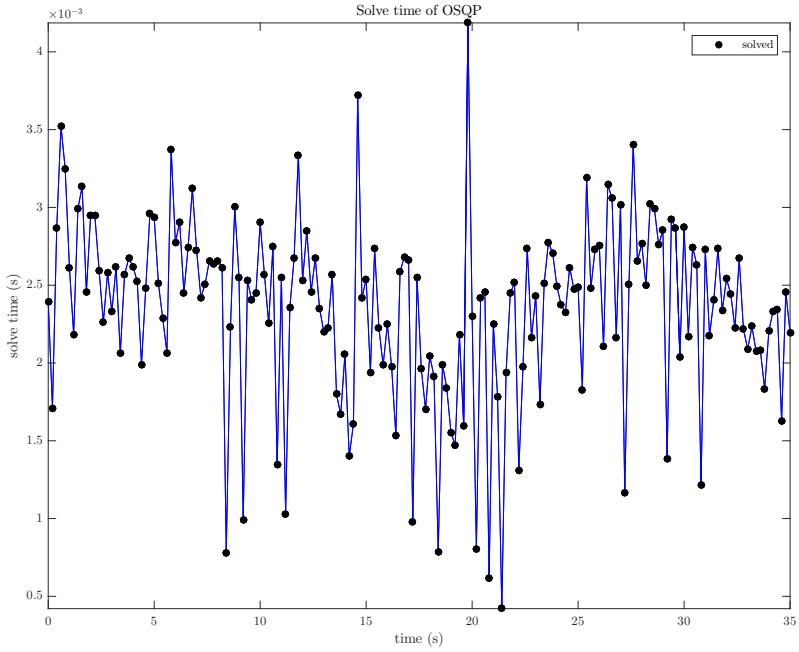

Figure D.5: OSQP solve time of stable MPC with frequency 5 Hz

Heidi-Irene Harvey Solue

Linear Model Predictive Control of a 3 DOF Helicopter

NTNU
Norwegian University of
Science and Technology