ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta aplikovaných věd
Katedra kybernetiky

BAKALÁŘSKÁ PRÁCE

Model-predictive control of helicopter model

PLZEŇ, 2021 ŠMÍD MATĚJ

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě bakalářskou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem bakalářskou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 11.5.2021 .....

vlastnoruční podpis

Acknowledgements
I would like to thank Ing. Martin Goubej PhD. for supervising my thesis and staying patient with my incessant questions regarding the work. I would also like to express my deepest gratitude towards my friends and family who supported me during the making of this thesis.

Poděkování
Chtěl bych poděkovat panu Ing. Matrinu Goubejovi PhD. za vedení mé práce a za to, že zachoval trpělivost i přes mé neustálé dotazy ohledně dalšího postupu. Také bych chtěl vyjádřit moji nehlubší vděčnost mým přátelům a rodině, kteří mě podporovali při mé tvorbě.

Abstract

The aim of this thesis is the design of an MPC controller tuned to a nonlinear mathematical model of a helicopter. This model is first modeled and identified using experiment data gathered on Humusoft CE150. The principle of MPC is described. The identified model is used to tune a MPC controller. Performance of this MPC controller is then compared with classical cascade PI-P control loop. Computational comparisons of different methods of solving the quadratic programming problems that arise from the MPC controller are also discussed.

Keywords: Model predictive control, Humusoft, 2DoF helicopter, Cascade control, quadprog, qpOASES, quadratic programming

Abstrakt

Cíl práce je vytvoření MPC regulátoru pro řízení matematického modelu helikoptéry. Tento model je vytvořen a identifikován za použití experimentálních dat naměřených na Humusoft CE150. Navržený MPC regulátor je porovnán s klasickou PI-P kaskádou pro řízení namodelované helikoptéry. Je zde také provedeno porovnání výpočetní náročnosti různých metod pro řešení problémů kvadratického programování, které je nutné řešit pro MPC regulátor.

Klíčová slova: Model predictive control, Humusoft, 2DoF helikoptéra, Kaskádní řízení, quadprog, qpOASES, kvadratické programování

Contents

1 Introduction **5**

2 Motivation **5**

3 Humusoft CE150 **6**

4 Linearization of a nonlinear pendulum system **6**
4.1 Pendulum model 6

5 Classical control **9**
5.1 PI control 9
5.2 PD control 11
5.3 PID control 13
5.4 PI control of velocity 14
5.4.1 Cascade control 16
5.5 Linear control conclusion 17

6 System identification **17**
6.1 Linear identification 17
6.2 Nonlinear identification 19
6.3 Zero-phase filter 21
6.3.1 FIR ZP filter 21
6.3.2 IIR ZP filter 21
6.3.3 Fourier analysis 22
6.4 Actuator model 24
6.4.1 Static characteristic 24
6.5 Measurement data identification 26
6.5.1 Finding the offset angle φ_{offset} 29
6.5.2 Statistical evaluation of identification accuracy 32
6.5.3 Useful linearized positions 33

7 MPC control **34**
7.1 MPC principle 34
7.1.1 Prediction 34
7.1.2 Optimal control 36
7.1.3 Unconstrained MPC 37
7.2 State estimator for input disturbance rejection 40
7.3 MPC constraints for helicopter model 41
7.4 Showcase of MPC control 41
7.5 Comparison to classical cascade control 44
7.5.1 Simulation results 48
7.6 qpOASES 55
7.6.1 Comparison between quadprog and qpOASES 55
7.6.2 Comparison conclusion 62

8 Conclusion **63**

1 Introduction

Model-predictive control is a control scheme utilising an internal model of the controlled plant to make predictions about its future states. It then uses these states together with past measured states to decide on the best possible control action by solving a quadratic programming problem. Model-predictive control has been demonstrated to be useful when it comes to control in industrial applications in the last 4 decades [1].

This thesis discusses the differences between a classical PID control scheme and Model Predictive Control. These different control laws are compared on a 2 Degree of Freedom helicopter model, whose mathematical model is devised in chapter 4 and 6. The aim is to show the possible limitations of both control schemes and decide on which is best suited for our model.

The first half of this paper explains the theory behind a SISO nonlinear model of the helicopter and its linearization, discusses the identification process, implements a zero-phase filtering mechanism for data gathered from an experiment and explains how the identification process can be used for gathering model parameters not calculated directly by the identification algorithm.

The second part of this paper then uses this identified model as the controlled plant. The principle of MPC control, of PID control, and of cascade PI control is explained. A cascade control loop is designed to control the identified model.
The MPC controller is designed and tuned using the linearized version of the identified model as its internal model. Methods of input disturbance rejection are discussed for both control schemes.

2 Motivation

The objective of this project is to decide whether the MPC control scheme is better suited for our helicopter model. MPC offers various advantages over classical PID control, namely, it allows us to set hard limits on the applied manipulated variable. This is especially useful when our available actuators are at risk of running in saturation when traditional PID control is used. It also allows us to limit the speed at which the manipulated variable changes. This allows us to limit the amount of nonlinear behaviour induced by the controller in the plant. Another advantage of MPC is its simple applicability for MIMO (Multi-Input Multi-Output) systems. An MPC controller can be implemented using only modification to computational algorithm but does not need any more complex parameter optimization to perform correctly. MIMO systems are notoriously hard to design using traditional PID control which makes MPC the ideal candidate for controlling such systems.
A great advantage of traditional PID control is its simplicity, both in the amount of parameters and in its computation. In contrast, MPC control is much more computationally intensive, forcing us to limit its sample rate and predictive capacities if we intend to use it in real time using slower hardware.

3 Humusoft CE150

Humusoft CE150 is a model helicopter designed for educational purposes. It is a MIMO system, with 2 actuators for pitch and yaw, and 2 sensor for each respective value. During this thesis we are only concerned about modeling the pitch axis, meaning we will be dealing with a SISO system only.

Figure 1: Model helicopter discussed in this thesis

Modeling of this exact helicopter model was discussed before in [2]. We chose to model the pitch of this helicopter using a second order nonlinear pendulum equation.

4 Linearization of a nonlinear pendulum system

In this section we will discuss a nonlinear model of a pendulum and its subsequent linearization about 5 different linearization points. This is done because the pendulum model behaves in a similar manner to the pitch axis part of the 2DoF helicopter model.

4.1 Pendulum model

The nonlinear model of a pendulum we will consists of a point mass m attached to a weightless arm of length l rotating around a pivot with friction b in a g. The pendulum is acted upon in the pivot with torque T.

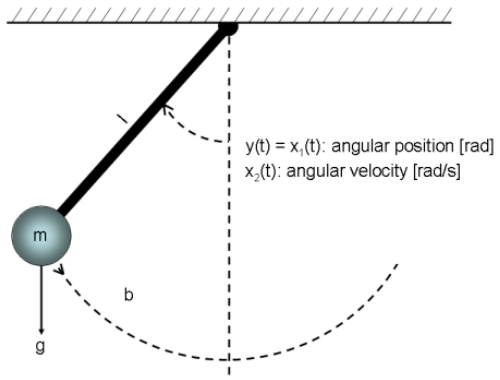


Figure 2: Nonlinear pendulum

The equation of motion of this system can be found using the Newton−Euler method. We determined them to be:

$$m\ddot{\varphi}l^2 + b\dot{\varphi} + mlgsin(\varphi) = T \tag{4.1}$$

The system is nonlinear thanks to the $sin(\varphi)$ term. Its phase space diagram is shown in figure 3.

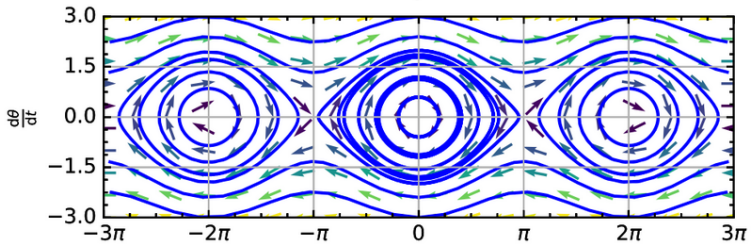


Figure 3: Phase space of an undampened pendulum

We can see that the pendulum oscillates around $0 \pm 2\pi k, k \in \mathrm{Z}$. This corresponds to its lowest point. We can also see that the pendulum is not able to stay still around $\pi \pm 2\pi k, k \in \mathrm{Z}$, this is because the there pendulum behaves in an unstable way. This matches our intuitive conceptions of a pendulum.

We will linearize this system at different points to better understand its behaviour. We will then also use the obtained linear models to test various classical control schemes.
For us to be able to perform linearization, the system has to be in a steady state. A system is in a steady state when its states are unchanging in time. Since the pendulum is a 2nd order system we only need to satisfy 2 equations. Namely $\ddot{\varphi}^* = 0$ and $\dot{\varphi}^* = 0$. When we apply these conditions to our equation of motion we get the equation for the steady state input T^*.

$$T^* = mlgsin(\varphi) \tag{4.2}$$

This is the torque required to keep the pendulum at a given angle φ. We can see from the steady state equations that we can linearize this system in any position. However, for our purposes, we will only perform this linearization at points $[0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$ since these cover all the principal eigenvalue positions.

To perform the linearization we first choose the states of our linear system $\mathbf{x}$ and their corresponding state equations. In our case these are $\ddot{\varphi}$ and $\dot{\varphi}$. We use basic algebra to separate the second time derivative from equation (4.1) and set the equation for the first time derivative. The input of out system u is the torque T.

$$\ddot{\varphi} = x_1 = -\frac{b\dot{\varphi} + mlgsin(\varphi) - T}{ml^2} \tag{4.3}$$

$$\dot{\varphi} = x_2 = \frac{\partial \varphi}{\partial t} \tag{4.4}$$

We also have to specify the relationship between the states of the system $\mathbf{x}$ and the output y. In a physical system this is often given by the sensor configuration, however, in our case we can choose this freely. For simplicity, we will use this output equation

$$y = \varphi \tag{4.5}$$

We structure the state and output equations as follows

$$\dot{\mathbf{x}} = \begin{bmatrix} \ddot{\varphi} \\ \dot{\varphi} \end{bmatrix} = \mathbf{f}(\mathbf{x}, T) \tag{4.6}$$

$$y = \varphi = \mathbf{h}(\mathbf{x}, T) \tag{4.7}$$

To obtain the state space representation in the form

$$\dot{\mathbf{x}} = \mathbf{A}\mathbf{x} + \mathbf{B}u$$
$$y = \mathbf{C}\mathbf{x}$$

of this nonlinear system we use the following formulae

$$\mathbf{A} = \frac{\partial \mathbf{f}(\mathbf{x}, T)}{\partial \mathbf{x}}|_{\ddot{\varphi}^*, \dot{\varphi}^*, T^*} \qquad \mathbf{B} = \frac{\partial \mathbf{h}(\mathbf{x}, T)}{\partial \mathbf{u}}|_{\ddot{\varphi}^*, \dot{\varphi}^*, T^*}$$
$$\mathbf{C} = \frac{\partial \mathbf{h}(\mathbf{x}, T)}{\partial \mathbf{x}}|_{\ddot{\varphi}^*, \dot{\varphi}^*, T^*} \qquad \mathbf{D} = \frac{\partial \mathbf{h}(\mathbf{x}, T)}{\partial \mathbf{u}}|_{\ddot{\varphi}^*, \dot{\varphi}^*, T^*}$$

The system linearized around an angle φ^* is as follows

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -\frac{gcos(\varphi^*)}{l} & -\frac{b}{ml^2} \end{bmatrix} \mathbf{x} + \begin{bmatrix} 0 \\ \frac{1}{ml^2} \end{bmatrix} u \tag{4.8}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix} \mathbf{x} \tag{4.9}$$

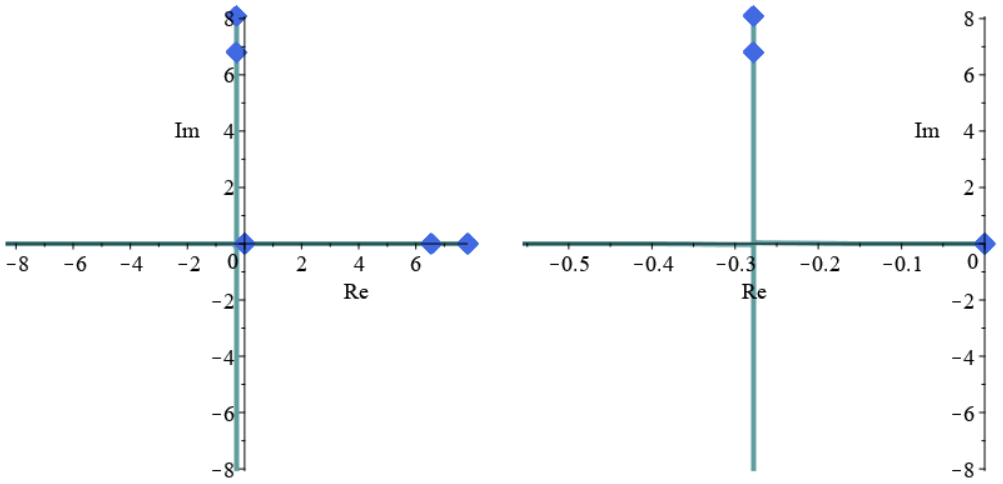
The system's poles lie in

$$\frac{-b + \sqrt{b^2 - 4m^2l^3gcos(\varphi^*)}}{2ml^2},$$
$$\frac{-b - \sqrt{b^2 - 4m^2l^3gcos(\varphi^*)}}{2ml^2}$$

To progress further we need to set the parameters m, l, b to particular values that approximate the real values of the model:

$$m = 0.4kg$$
$$l = 0.15m$$
$$b = 0.005$$

For these values we created a plot of the possible pole locations for angles φ^* from 0 to π. The blue diamonds in the plot are the location of one of the poles for $\varphi^* = [0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$.



(a) Pole map for the linearized system (b) Close up of the root locus zoomed at stable values

Figure 4: Root locus of the various linearized systems

The poles for $\varphi^* = 0$ start as complex conjugates at $-0.278 \pm 8.082i$ and as φ^* increases their imaginary part gets closer to zero, until $\varphi^* = 1.571$ where the two poles meet at $[-0.278, 0]$ and they start moving in opposite direction on the real axis. At precisely $\varphi^* = \frac{\pi}{2}$, the system starts exhibiting the behaviour of an integrator. For $\varphi^* > \frac{\pi}{2}$, one of the poles crosses over into the right hand half of the complex plane. This makes the system unstable. The real part of the complex conjugate poles is $Re(p) = -\frac{b}{2ml^2} = -0.277$. The system's gain is $K = \frac{1}{ml^2} = 111$

5 Classical control

In this section, we will design linear controllers using classical design methods. We will test these controllers on the linearized systems discussed in the previous section.

5.1 PI control

The most widely used control scheme is a PI controller in a feedback loop around the controlled plant. We will design a PI controller for φ^* angles: $[0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$ using the following form:

$$C = K_p(1 + \frac{1}{T_i s}) \tag{5.1}$$

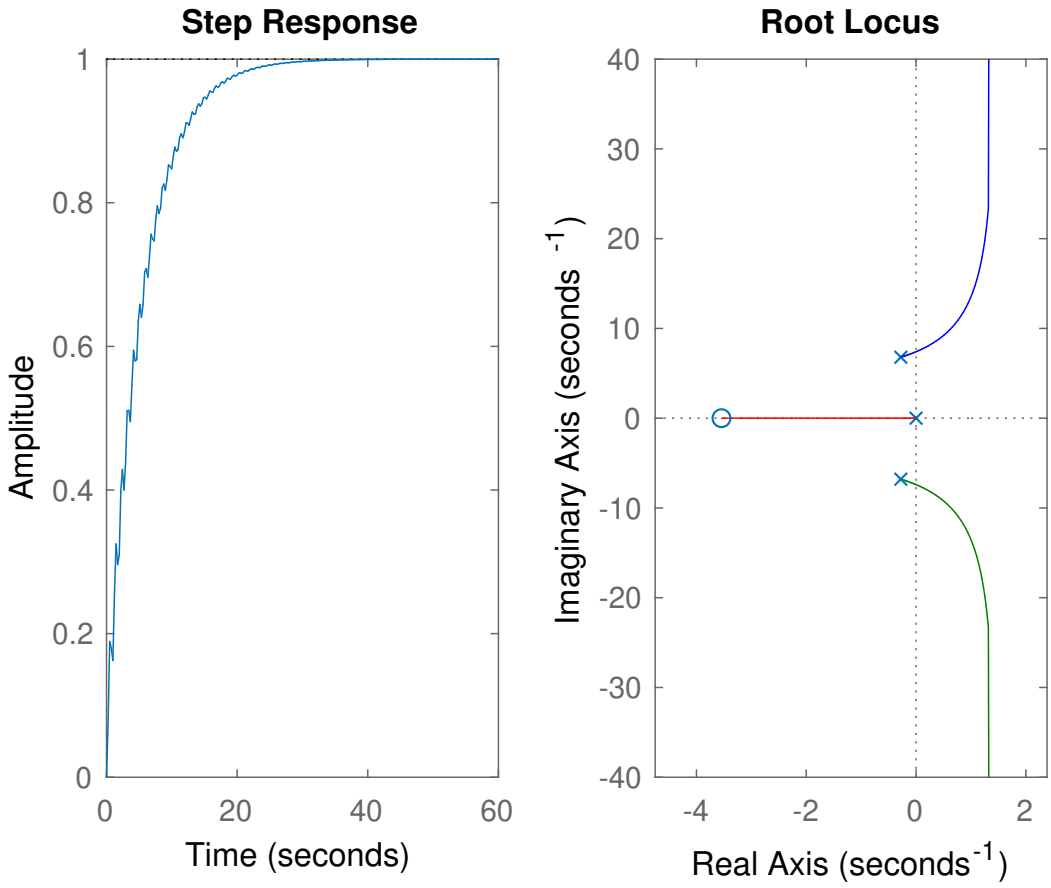


Figure 5: Closed loop step response and the root locus plot of the system linearized around $\varphi^* = \frac{\pi}{4}$

In theory, a PI controller in a negative feedback loop is capable of stabilising the pendulum model linearized around any angle φ^*. However, for any angles between $\frac{\pi}{2}$ and $\frac{3\pi}{2}$ the system cannot be controlled with a satisfying degree of robustness (Phase margin $< 2°$) , unless the dissipation factor b is disproportionately large. As can be seen from the root locus plot. A PI controller is well suited for control of the stable versions of the system (linearized around $\varphi \leq \frac{\pi}{2}$).

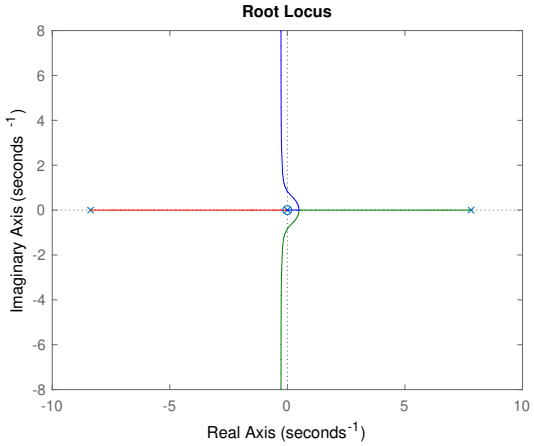


Figure 6: Root locus plot of the PI controlled system linearized around $\varphi^* = \pi$

The linearized system pole's are almost symetrical about the imaginary axis, only being offset to the left on the real axis by $\frac{-b}{2ml^2}$. Because the PI controller adds one zero and one pole to the open loop system, the location of the 2 free poles in the root locus tends asymptotically towards infinity with the angles $\frac{\pi}{2}$ and $\frac{3\pi}{2}$. The intersection point of asymptotes on the real axis is called the centroid. We can calculate the location of the centroid α using this formula:

$$\alpha = \frac{\sum Re(poles) - \sum Re(zeros)}{numberofpoles - numberofzeros}$$

In our case the sum of real poles is the offset factor $\frac{-b}{2ml^2}$. The only zero in our open loop system is the one added by the PI controller. To achieve the best robustness in stability possible, we want to choose the location of the zero such that the centroid is as far left as possible. The best case scenario for this is placing the zero in the origin. However, this will will only allow us to move the centroid to the offset factor. This can be expressed using this equation:

$$\lim_{zero_{PI} \to 0} \alpha = \frac{-b}{2ml^2}$$

From this, we can see that using a PI controller for control of an unstable pendulum system is not advisable.

5.2 PD control

PD controller with a filtering term is able to move the poles of the controlled plant more freely than a PI controller, however, since it lacks an integrator pole, it does not guarantee the error term will reach 0 when following a constant reference value. We will design a PD controller for φ^* angles: $[0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$ using the following form:

$$C = K_p(1 + \frac{T_d\, s}{\frac{T_d}{N} s + 1}) \tag{5.2}$$

Using the same intuition about the root locus plots from the previous section we can study if this type of controller is viable for this system.

A PD controller adds one negative zero and one even more negative pole to the open loop system. This means that the root locus has 2 free poles that will asymptotically tend towards infinity with angles $\frac{\pi}{2}$ and $\frac{3\pi}{2}$ from the centroid. This centroid can be located using the same formula as in the previous section. The pole added by the filtered term of the PD controller is farther to the left on the complex plane than the zero. From this we can conclude that this controller can shift the centroid of this open loop system into the left hand plane, thus stabilising the closed loop system.

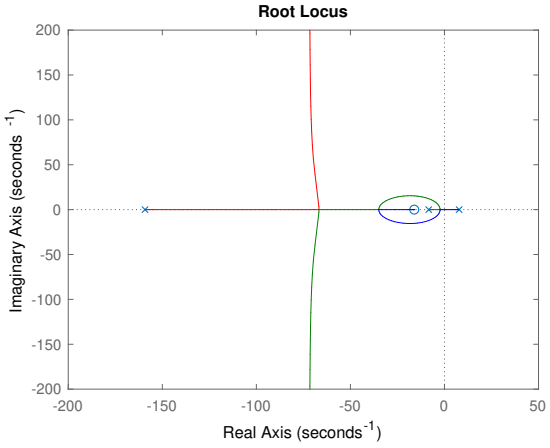


Figure 7: Root locus plot of the PD controlled system linearized around $\varphi^* = \pi$

However, while using the PD controller, the open loop system has no integrator (with the exception of the system linearized around $\varphi^* = \frac{\pi}{2}$). This means that the closed loop system lacks the intrinsic ability to follow a step signal without a constant error term. This can be partly mitigated by introducing a open loop amplification term, which will move the closed loop amplification of the control loop to 1. We will solve this issue by introducing an integrator term into the controller, thus making it a PID controller. This will be discussed in the next section.

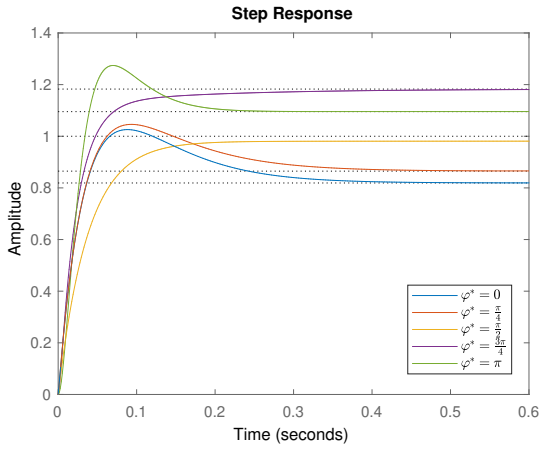


Figure 8: Closed loop step response of PD controlled systems

5.3 PID control

A PID controller with a filtering term combines the strenghts of PI and PD controllers. It enables the feedback loop to reach zero error when following a constant reference, but also allows us to move the poles of the controlled plant more freely. We will design a PID controller for φ^* angles: $[0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$ using the following form:

$$C = K_p(1 + \frac{1}{T_i s} + \frac{T_d s}{\frac{T_d}{N} s + 1}) \tag{5.3}$$

The root locus plots we get by using a PID controller are very similar to the ones we get by using a PD controller. This is because the PID controller with a filtering term adds one pole and one zero on top of the pole and zero introduced by a PD controller. This means that the open loop system still has 2 more poles than zeros and thus the root locus forms two arms which asymptotically tend towards infinity with angles $\frac{\pi}{2}$ and $\frac{3\pi}{2}$ from the centroid. The location of the centroid can be set using the controller variables to be in the left hand plane. Therefore we can conclude that a PID controller can stabilize these systems.

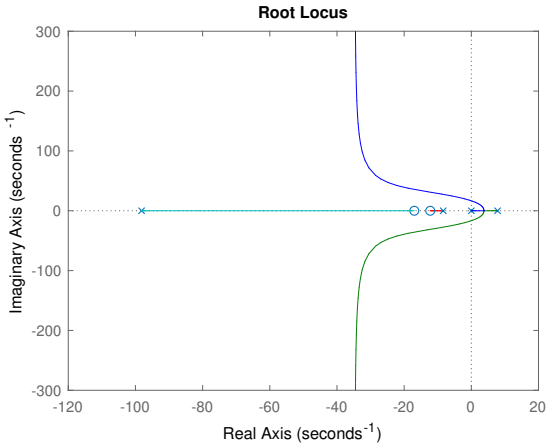


Figure 9: Root locus plot of the PID controlled system linearized around $\varphi^* = \pi$

Thanks to the pole located in the origin introduced to the open loop system by the PID controller, the closed loop system is capable of following a step signal without a constant error term.

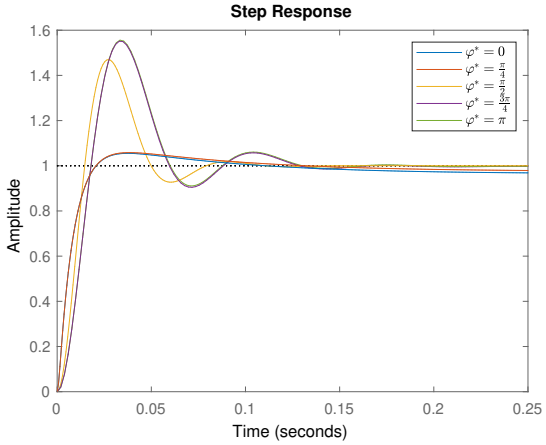


Figure 10: Closed loop step response of PID controlled systems

5.4 PI control of velocity

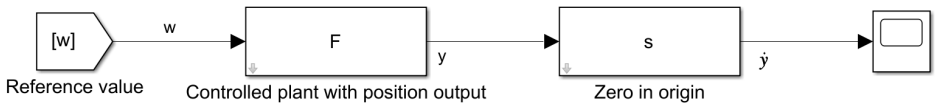
We can change our controlled model of the plant by introducing a derivator (zero in origin) after the plant output. This effectively mimics how the plant would behave if we were to measure the pendulum velocity instead of its angle. This is equivalent to changing the output matrix $\mathbf{C}$ in equation (4.9) to

$$\mathbf{C} = \begin{bmatrix} 0 & 1 \end{bmatrix} \tag{5.4}$$

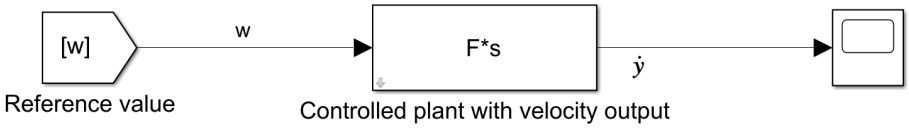
This approach allows us to introduce an additional zero into the loop, similar to what would happen when using a PD controller. We can then use a PI controller to control the angular velocity of the pendulum plant. We will design a PI controller for φ^* angles: $[0, \frac{\pi}{4}, \frac{\pi}{2}, \frac{3\pi}{4}, \pi]$ using the following form:

$$C = K_p(1 + \frac{1}{T_i s}) \tag{5.5}$$

The pendulum transfer functions are set up in such a way that their output is the position of the pendulum in time (φ). Since angular velocity is the derivative of position we can simply add a transfer function with a zero in the origin in series after the pendulum transfer function. We can combine these two transfer functions into one using block diagram algebra.



Into



The resulting transfer function shares the dynamics of the original system, but its output is the angular velocity of the pendulum $\dot{\varphi}$.

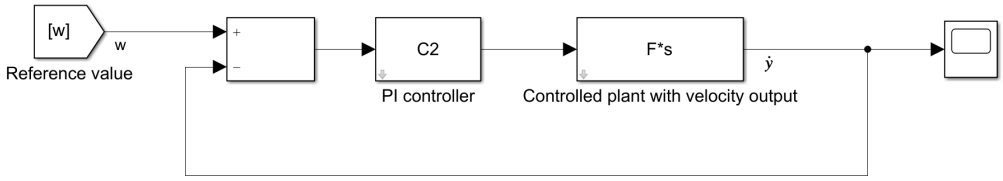


Figure 11: Block algebra diagram of $\dot{\varphi}$ plant controlled by a PI controller

The PI controller adds a pole to the origin. This cancels out the zero added to the origin by changing the system output to velocity. The resulting open loop system has 2 poles and one zero. This changes the shape of the root locus plot in comparison to the plots in the previous section where we controlled the location. As long as the zero added by the PI controller is in the left half plane, the rightmost pole will tend toward the location of the zero, while the leftmost pole will tend towards $-\inf$ on the real axis. This means that we can make all of our linear plants stable.

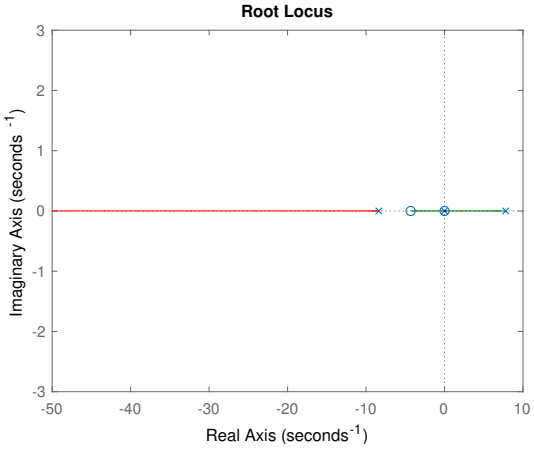


Figure 12: Root locus plot of the PI controlled $\dot{\varphi}$ plant linearized around $\varphi^* = \pi$

However, because the integrator introduced by the PI controller cancels out with the zero in the origin of the linear plant, the closed loop system lacks the ability to follow a constant reference signal without a constant error term. This can be seen in figure 13.

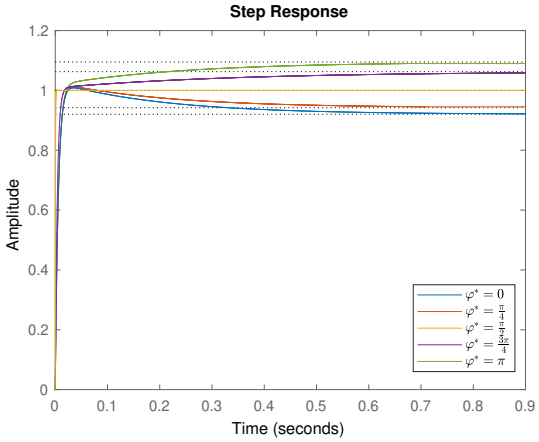
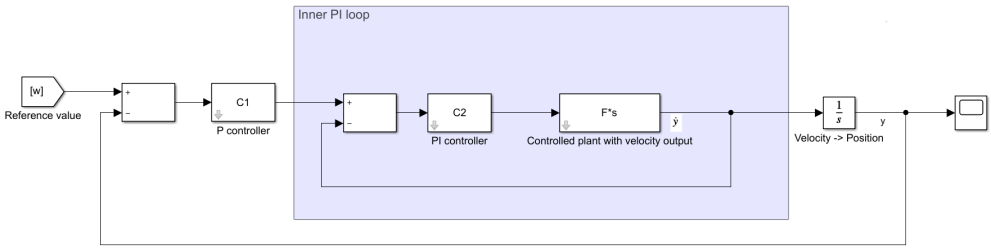


Figure 13: Step response of $\dot{\varphi}$ using PI control

5.4.1 Cascade control

To use the designed PI velocity control system to control the location of the pendulum, we can place an integrator after it and use this as feedback value in a negative feedback loop. This will convert the output to the angle φ, instead of the angular velocity $\dot{\varphi}$. The outer feedback loop will also ensure a zero steady state error term when following a constant reference signal. We can also add a P controller to the outer feedback loop to change the aggression of the control loop.



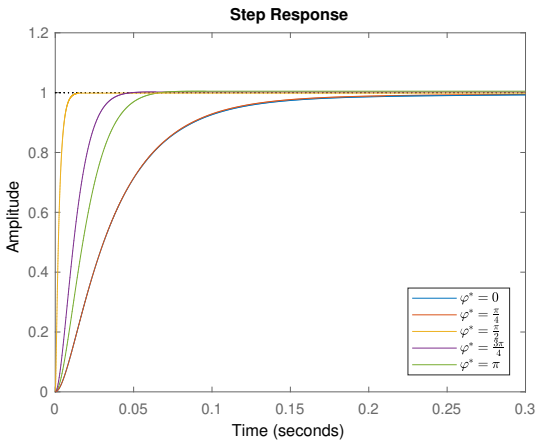


Figure 14: Step response of φ using cascade control

5.5 Linear control conclusion

When we compare all the linear control methods we used to stabilise the pendulum system, we can see that both the PID controller for the position, and the cascade control scheme, outperform the possible closed loop dynamics that are achievable with only a PI controller. With the cascade control being clearly better than the designed PID controller.

6 System identification

In order to tune the controllers correctly, we need to know the exact parameters of the system. In other words we need to identify the system. The nonlinear ordinary differential equation which we will use to model our system is

$$ml^2\ddot{\varphi} + b\dot{\varphi} + mlgsin(\varphi) = ku \tag{6.1}$$

The right hand side of the equation represents the torque applied to the pendulum by a motor. We chose to model this as constant gain k times input voltage u. We divide the equation by ml^2 to reduce the number of different parameters used.

$$\ddot{\varphi} + \frac{b}{ml^2}\dot{\varphi} + \frac{g}{l}sin(\varphi) = \frac{k}{ml^2}u \tag{6.2}$$

We can substitute the parameters used in the rearranged equation such that $a_1 = \frac{b}{ml^2}$, $a_0 = \frac{g}{l}$, $b_0 = \frac{k}{ml^2}$.

$$\ddot{\varphi} + a_1\dot{\varphi} + a_0sin(\varphi) = b_0u \tag{6.3}$$

To fully identify the parameters of the model, we only need to identify a_1, a_0 and b_0. This simplifies the identification, but doesn't allow us to identify the physical parameters of the system m, l, b, k.

6.1 Linear identification

One of the possible approaches to the identification problem is to create a regression model linear in both the parameters and the system states. To do this we linearize the nonlinear ODE (6.3) around a equilibrium point φ^*.

$$\dot{\mathbf{x}} = \begin{bmatrix} 0 & 1 \\ -a_0cos(\varphi^*) & -a_1 \end{bmatrix}\mathbf{x} + \begin{bmatrix} 0 \\ b_0 \end{bmatrix}u \tag{6.4}$$

$$y = \begin{bmatrix} 1 & 0 \end{bmatrix}\mathbf{x} \tag{6.5}$$

Next, we convert the plant into a transfer function and then discretize it using zero-hold method with sample time T. We get a transfer function in the z-domain

$$G(z) = \frac{Y(z)}{U(z)} = \frac{B_1z + B_0}{A_2z^2 + A_1z + A_0}$$

Where

$$B_1 = -b_0(4e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_0cos(\varphi) + 4e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_0cos(\varphi)$$
$$-8a_0cos(\varphi)e^{-Ta_1} + e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1\sqrt{a_1^2-4a_0cos(\varphi)} - e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1^2$$
$$-e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1\sqrt{a_1^2-4a_0cos(\varphi)} - e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1^2 + 2a_1^2e^{-Ta_1})$$
$$B_0 = -b_0(4e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_0cos(\varphi) + 4e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_0cos(\varphi)$$
$$-e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1\sqrt{a_1^2-4a_0cos(\varphi)} - e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1^2 +$$
$$e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1\sqrt{a_1^2-4a_0cos(\varphi)} - e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}a_1^2 - 8a_0cos(\varphi) + 2a_1^2)$$
$$A_2 = 2e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2}(-a_1^2 + 4a_0cos(\varphi))a_0cos(\varphi)$$
$$A_1 = 2(-e^{((-a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2} - e^{-((a_1+\sqrt{a_1^2-4a_0cos(\varphi)})T)/2})(-a_1^2 + 4a_0cos(\varphi))a_0cos(\varphi)$$
$$A_0 = 2(-a_1^2 + 4a_0cos(\varphi))a_0cos(\varphi)$$

To convert this z-domain transfer function into the discrete time domain, we perform the inverse Z-transform.

$$A_2y(k) + A_1y(k-1) + A_0y(k-2) = B_1u(k-1) + B_0u(k-2)$$

We rearrange this difference equation to separate the $y(k)$ term.

$$y(k) = -\frac{A_1}{A_2}y(k-1) - \frac{A_0}{A_2}y(k-2) + \frac{B_1}{A_2}u(k-1) + \frac{B_0}{A_2}u(k-2)$$

This is the equation we can use to build the linear regression model to identify the system parameters using measurement data. The equation can be expressed in matrix form for l data samples

$$\mathbf{y} = \mathbf{\Phi\Theta} + \boldsymbol{\epsilon}$$

Where

$$\mathbf{y} = \begin{bmatrix} y(k) \\ y(k+1) \\ \vdots \\ y(k+l) \end{bmatrix},$$

$$\mathbf{\Phi} = \begin{bmatrix} -y(k-1) & -y(k-2) & u(k-1) & u(k-2) \\ -y(k) & -y(k-1) & u(k) & u(k-1) \\ \vdots & \vdots & \vdots & \\ -y(k+l-1) & -y(k+l-2) & u(k+l-1) & u(k+l-2) \end{bmatrix},$$

$$\mathbf{\Theta} = \begin{bmatrix} \frac{A_1}{A_2} \\ \frac{A_0}{A_2} \\ \frac{B_1}{A_2} \\ \frac{B_0}{A_2} \end{bmatrix}, \boldsymbol{\epsilon} = \begin{bmatrix} \epsilon(k) \\ \epsilon(k+1) \\ \vdots \\ \epsilon(k+l) \end{bmatrix}$$

We define the least squares criterion for this system of equations as

$$J(\mathbf{\Theta}) = \boldsymbol{\epsilon}^T(\mathbf{\Theta})\boldsymbol{\epsilon}(\mathbf{\Theta})$$

The vector $\mathbf{\Theta}$ that minimizes this criterion is the optimal vector of parameters $\mathbf{\Theta}^*$

$$\mathbf{\Theta}^* = argmin[J(\mathbf{\Theta})]$$

Unfortunately, this approach yields sub par results for our system and because of the complicated transform functions from parameters the model a_1, a_0, b_0 to the parameters of the regression model $\mathbf{\Theta}$ it would be very difficult to find the inverse transform to identify the model. Because of these reasons, we will not pursue this method here any further.

6.2 Nonlinear identification

Another approach to the identification problem is to create a regression model that will be linear in its parameters, but will be nonlinear in the system states. To do this we need to perform an experiment with the helicopter model and obtain the angle φ, velocity $\dot{\varphi}$, acceleration $\ddot{\varphi}$ and input u data. We are able to directly measure the angle φ and the input u is chosen by us. However, we are not able to measure the velocity and acceleration directly, owing to the fat that the Humusoft CE150 helicopter lacks the appropriate sensors, we will have to compute them numerically from the angle signal.

We will compute them using the first and second central differences

$$\dot{y}(k) = \dot{\varphi}(k) = \frac{\varphi(k+1) - \varphi(k-1)}{2T_S} \tag{6.6}$$

$$\ddot{y}(k) = \ddot{\varphi}(k) = \frac{\varphi(k+1) - 2\varphi(k) + \varphi(k-1)}{T_s^2} \tag{6.7}$$

We then created a regression model from the ODE in (6.3), thanks to making a substitution into discrete time domain with $kT_s = t$, where T_s is the sample time.

$$\ddot{y}(k) = -a_1\dot{y}(k) - a_0 sin(y(k)) + b_0 u(k) \tag{6.8}$$

This difference equation expands into multiple time samples which can be expressed using the matrix form

$$\ddot{\mathbf{y}} = \mathbf{\Phi\Theta} + \boldsymbol{\epsilon} \tag{6.9}$$

where

$$\ddot{\mathbf{y}} = \begin{bmatrix} \ddot{y}(k) \\ \ddot{y}(k+1) \\ \vdots \\ \ddot{y}(k+l) \end{bmatrix}, \tag{6.10}$$

$$\mathbf{\Phi} = \begin{bmatrix} -\dot{y}(k) & -sin(y(k)) & u(k) \\ -\dot{y}(k+1) & -sin(y(k+1)) & u(k+1) \\ \vdots & \vdots & \vdots \\ -\dot{y}(k+l) & -sin(y(k+l)) & u(k+l) \end{bmatrix} \tag{6.11}$$

This regression model is linear in the parameters $\mathbf{\Theta}$. This is a requirement for the method of identification we will use.

To obtain our parameters $\mathbf{\Theta}$ we will use the least squares criterion defined as

$$J(\mathbf{\Theta}) = \boldsymbol{\epsilon}^T(\mathbf{\Theta})\boldsymbol{\epsilon}(\mathbf{\Theta})$$

The vector $\mathbf{\Theta}$ which minimizes this criterion is the optimal vector of parameters $\mathbf{\Theta}^*$

$$\mathbf{\Theta}^* = argmin[J(\mathbf{\Theta})]$$

The vector $\mathbf{\Theta}$ is made up of the parameters of the ODE

$$\mathbf{\Theta} = \begin{bmatrix} a_1 \\ a_0 \\ b_0 \end{bmatrix}$$

The disadvantage of this method of identification is that the differences $\ddot{\varphi}(k)$ and $\dot{\varphi}(k)$ are calculated from the location data φ and any measurement noise present will be multiplied in the differences, thanks to the property of derivation to amplify high frequency noise. This can be remedied by filtering the measurement data of any noise. However, standard filtering methods shift the phase of the signal which would make the signal unusable for identification purposes.

To solve this, we will use a method of digital signal filtering called zero-phase filtering.

6.3 Zero-phase filter

Zero-phase (ZP) filter is a a kind of digital filter in which the phase shift introduced by the filter is 0 on all frequencies. In other words, this kind of filter only changes the amplitude of the filtered signal. This property makes ZP filters non-causal and so they cannot be directly used in real time application. Thankfully this non-causality is not an issue, because the data gathering for the identification is done before the filtering, therefore the filter in time step k has access to the future, as well as the past measurement signal. This makes the ZP filters ideal for our purposes. We will discuss 2 different methods of implementing a ZP filter.

6.3.1 FIR ZP filter

Finite impulse response zero-phase filter is a digital filter whose impulse response function is finite, has a odd number of time steps and is symmetric about time step 0. An example of this filter is

$$H_{FIR}(z) = z^{-1} + 2 + z \tag{6.12}$$

It can be seen that this filter is non-causal, because it uses future signal to compute the filtered signal at the current time step. The number of terms in the impulse response function has to be odd to achieve the symmetry needed for the filter to be zero-phase.

This filter can be generally written as

$$H_{FIR}(z) = \sum_{k=-N}^{N} b_k z^{-k} \tag{6.13}$$

where the symmetry is ensured by the fact that $b_{-k} = b_k$.

6.3.2 IIR ZP filter

Another way to implement a zero-phase filter is to use a causal infinite impulse filter to filter a signal, then to filter the filtered signal in reverse time order. Finally reverse the time order of this twice filtered signal once more. The result is a signal, whose amplitude is amplified by the square of the amplitude characteristic of the used IIR filter, and whose phase shift is zero.

For a causal IIR filter

$$H_C(z) = \sum_{k=0}^{2N} a_k z^{-k} \tag{6.14}$$

the filtering is reverse time order using the same IIR filter is non causal and written as

$$H_C(z^{-1}) = \sum_{k=0}^{2N} a_k z^{k} \tag{6.15}$$

The composite filtering as described above can be expressed as

$$H_{NC}(z) = H_C(z^{-1}) H_C(z) \tag{6.16}$$

The properties of this type of filtering can be seen in Fourier space ($z \to e^{i\omega}$)

$$H_{NC}(e^{i\omega}) = H_C(e^{-i\omega})H_C(e^{i\omega}) = \tag{6.17}$$
$$= \overline{H_C}(e^{i\omega})H_C(e^{i\omega}) = \tag{6.18}$$
$$= |H_C(e^{i\omega})|^2 \tag{6.19}$$

H_{NC} is strictly real and greater than 0 in Fourier space and therefore does not introduce any phase shift into the filtered signal.
A IIR ZP filter can be implemented using the MATLAB function *filtfilt()* with a standard IIR filter.

6.3.3 Fourier analysis

To determine the appropriate lowpass frequency for our filter we looked at the frequency spectrum of our signal obtained using MATLAB's $fft()$ function. This signal is sampled with the sampling frequency $f_s = 50Hz$. The *Nyquist–Shannon sampling theorem* states that frequencies higher that the Nyquist frequency $f_N = f_s/2$ cannot be reconstructed. Our Fourier transform therefore only considers frequencies up to the Nyquist frequency $25Hz$.

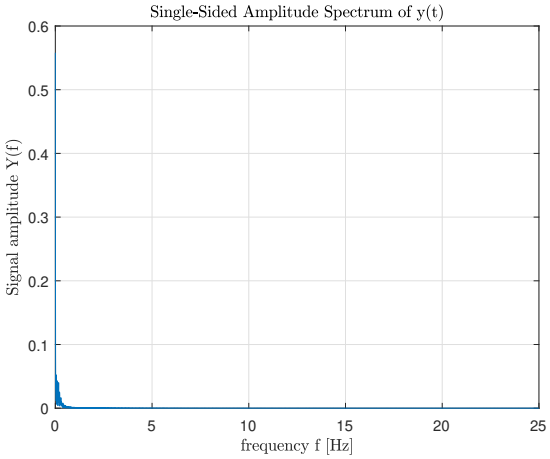


Figure 15: Fourier spectrum of the φ value obtained from the experiment

The experiment was measured in such a way as not to introduce any high frequency information into the data. We can use this to help us determine the appropriate lowpass frequency for filtering.

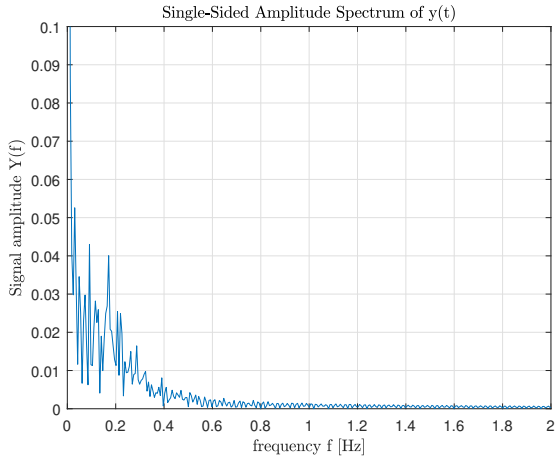


Figure 16: Fourier spectrum of the φ value obtained from the experiment zoomed on the relevant frequency range

Most of the energy of the signal lies on the frequencies from 0 to $0.5Hz$. We will assume that anything over $1Hz$ is noise which should be removed prior to continuing with the identification.

For this removal we will use a FIR lowpass filter of 100th order with lowpass frequency $1Hz$.

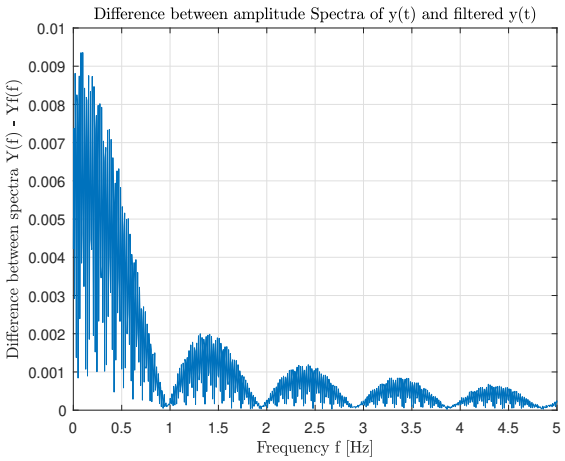


Figure 17: Energy removed from the data by the filtering process

As can be seen from the figure, the filter does not remove energy evenly across the frequency range. Most importantly it also filters before our lowpass frequency. This fact can however be ignored, since the relative reduction on low frequencies is minor compared to the relative reduction on the higher frequencies.

6.4 Actuator model

The pitch angle of the helicopter model as described above is

$$m\ddot{\varphi}l^2 + b\dot{\varphi} + mlgsin(\varphi) = T \tag{6.20}$$

This equation assumes a input torque T, we however cannot directly influence T. Instead we can only influence it indirectly by applying a voltage u on a actuator unit. The actuator itself has its own dynamic behaviour described by

$$L\frac{di(t)}{dt} + Ri(t) = u(t) - k_e\omega(t) \tag{6.21}$$

This first order linear ODE describes the dynamics of the electrical circuit, ie. the way the input voltage $u(t)$ influences the electrical current $i(t)$ and how this direct relationship is hampered by the rotational velocity of the blades of the helicopter $\omega(t)$. The constant k_e is the electromotoric constant of the motor. L is the inductance of the motor. R is the electrical resistance in the circuit. $\omega(t)$ is measured in rad/s

$$J\frac{d\omega(t)}{dt} = k_t i(t) - \alpha\omega|\omega| \tag{6.22}$$

This first order nonlinear ODE describes the dynamic behaviour of how the circuit current $i(t)$ affects the rotational velocity $\omega(t)$. This relationship is again hampered by the rotational velocity $\omega(t)$, this time however, the $|x|$ denotes the absolute value of x. $\omega|\omega|$ has the is identical to ω^2 for positive values of ω, but has the characteristic of $-\omega^2$ for the negative values. J is the moment of inertia of the blades. k_t is the momentum constant of the motor ($i(t) \rightarrow$ generated T). α is the parameter describing the changing resistance of the blades based on their rotational velocity.

$$F = \beta\omega|\omega| \tag{6.23}$$

This nonlinear static equation describes how the rotational velocity $\omega(t)$ generates lift force F. This force creates a torque T about the centre of mass of the helicopter. Assuming the simplified pendulum model, this torque is calculated as $T = F \cdot l$ where l is the length of the pendulum arm.

6.4.1 Static characteristic

The above described motor equations are dynamic in nature. We wanted to see the simplified relationship between the input voltage u_0 and the output force F_0 in steady-state. To do this we set all the derivatives to 0 to obtain the steady-state equations

$$Ri_0 = u_0 - k_e\omega_0 \tag{6.24}$$
$$0 = k_t i_0 - \alpha\omega_0|\omega_0| \tag{6.25}$$
$$F_0 = \beta\omega_0|\omega_0| \tag{6.26}$$

And algebraically combined the first two into

$$\alpha\omega_0|\omega_0|+\frac{k_e k_t}{R}\omega_0 = \frac{k_t}{R}u_0 \tag{6.27}$$

The solution of this equation is

$$\omega_0 = \begin{cases} \frac{\frac{k_e k_t}{R} - \sqrt{(\frac{k_e k_t}{R})^2 - 4\frac{k_t}{R}u_0}}{2\alpha} & u_0 \leq 0 \\ \frac{-\frac{k_e k_t}{R} + \sqrt{(\frac{k_e k_t}{R})^2 + 4\frac{k_t}{R}u_0}}{2\alpha} & u_0 \geq 0 \end{cases} \tag{6.28}$$

Therefore, the static function $F_0(u_0)$ is obtained by substituting this solution into the output equation

$$F_0(u_0) = \beta\omega_0|\omega_0| \tag{6.29}$$

We are only interested in the positive branch of this characteristic (positive voltage generating positive force) i.e.

$$\omega_0^+ = \frac{-\frac{k_e k_t}{R} + \sqrt{(\frac{k_e k_t}{R})^2 + 4\frac{k_t}{R}u_0}}{2\alpha} \quad u_0 \geq 0 \tag{6.30}$$

An example plot of this characteristic for parameters: $[k_e = 1, k_t = 1, R = 1, \alpha = 1, \beta = 1]$.

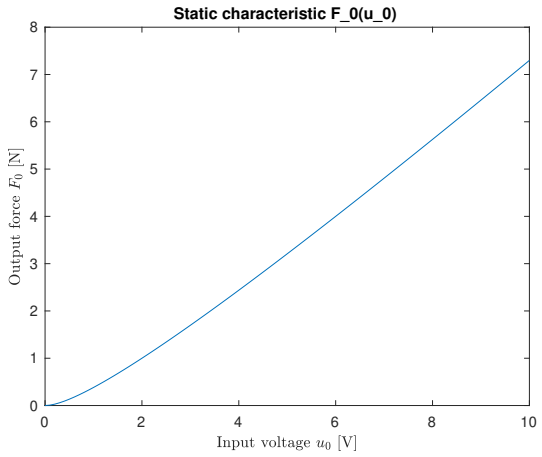


Figure 18: Static characteristic for arbitrarily chosen physical parameters

This plot exhibits a quadratic behaviour around the origin, but quickly moves into an almost linear shape.

6.5 Measurement data identification

An experiment was performed on the helicopter model.Measurement data from the physical model helicopter was purposefully taken with care to not excite the dynamic response of the actuator. The measurement was taken over approximately 120 seconds with $20ms$ sample time.

Only measurements available to us are the measured angle of pitch φ_{meas} from an optical sensor [°] and the input voltage applied to the motor u. This voltage is expressed as a fraction of the maximum power the motor is capable of providing with 0 being turned off and 1 being full power. The optical sensor used for this measuring has a quantization step of 0.36°. This quantization intruduces noise into our measurement, which is amplified when we calculate the first and second order derivatives to obtain the velocity and the acceleration. This noise is then filtered out by using a FIR filter to smooth this data out and to remove the higher frequency noise.

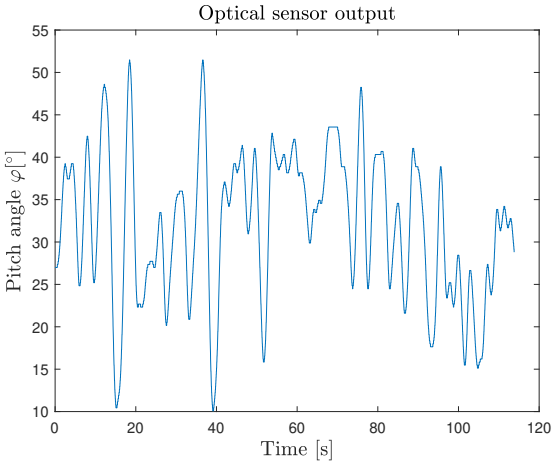


Figure 19: Pitch angle data used for identification

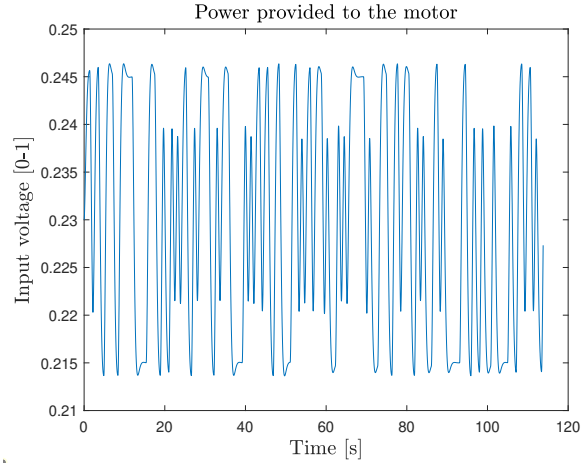


Figure 20: Input voltage data used for identification

The measured pitch angle φ_{meas} is not the true pitch angle measured from the vertical, rather only the angle measured from an unknown offset angle φ_{offset}. To correctly identify the whole model we need to find this angle as will be discussed below. To progress further

we have to set the offset angle to an arbitrary value and add it to the pitch angle data. For our purpose we can choose to set it as $\varphi_{offset} = 0$.

The angles φ_{meas} are used to calculate the angular velocity $\dot{\varphi}$ and the angular acceleration $\ddot{\varphi}$. For these we no longer need to consider the unknown offset angle φ_{offset} since the calculations used to find them cancels the offset angle out.

$$\dot{\varphi} = \frac{\varphi(k+1) - \varphi(k-1)}{2T} \tag{6.31}$$

$$\ddot{\varphi} = \frac{\varphi(k+1) - 2\varphi(k) + \varphi(k-1)}{T^2} \tag{6.32}$$

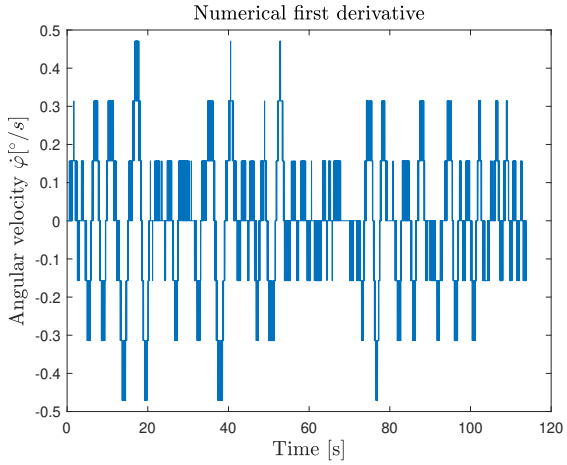


Figure 21: Calculated first derivative used for identification

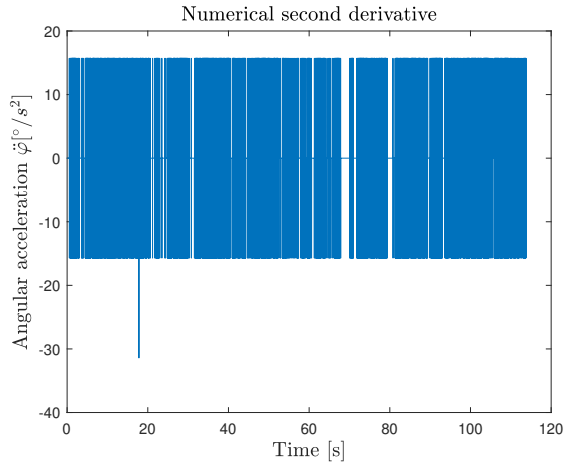


Figure 22: Calculated second derivative used for identification

As was said earlier, the numerical derivation amplifies the limited quantized values. This is solved in the next step where the data is filtered through a lowpass FIR filter of order 100 to remove any signal with frequency above $1Hz$. This also smooths out the data.

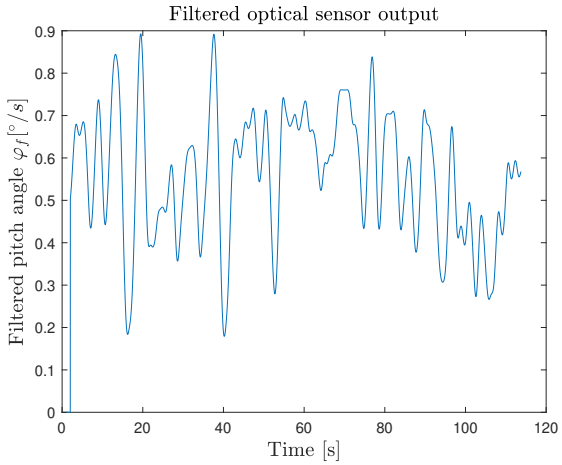


Figure 23: Filtered pitch angle used for identification

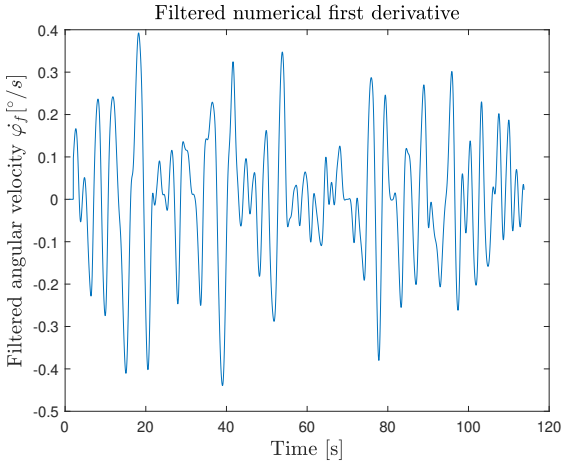


Figure 24: Filtered first derivative used for identification

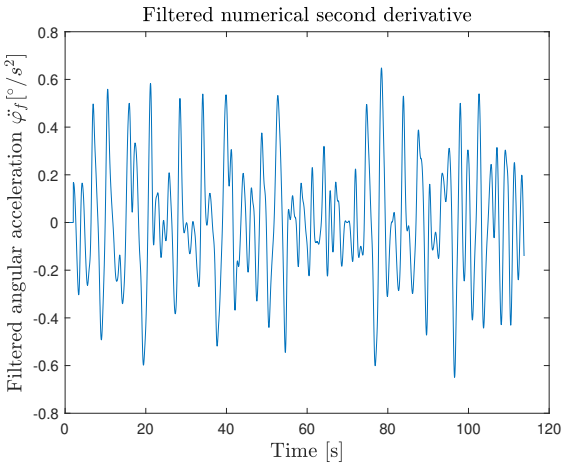


Figure 25: Filtered second derivative used for identification

The filtering algorithm shortens the filtered data by half of the filter order at the beginning and another half at the end. In our case of a 100 order FIR filter this equals to one second at the beginning and 1 second at the end of data loss. This is not significant and does not affect the identification process in any meaningful way.

Using the data filtered this way and the input voltage we measured we begin the identification part proper. We build the matrix of regressors

$$\mathbf{\Phi} = \begin{bmatrix} -\dot{\varphi}_f(k) & -sin(\varphi_f(k)) & u(k) & \\ -\dot{\varphi}_f(k+1) & -sin(\varphi_f(k+1)) & u(k+1) & \\ \vdots & \vdots & \vdots & \vdots \\ -\dot{\varphi}_f(k+l) & -sin(\varphi_f(k+l)) & u(k+l) & \end{bmatrix} \tag{6.33}$$

And calculate its pseudo-inverse as

$$\mathbf{\Phi}^{\dagger} = (\mathbf{\Phi}^T \cdot \mathbf{\Phi})^{-1} \cdot \mathbf{\Phi} \tag{6.34}$$

And obtain the identified parameters of the model Θ as

$$\Theta = \mathbf{\Phi}^{\dagger} \cdot \ddot{\varphi}_f \tag{6.35}$$

6.5.1 Finding the offset angle φ_{offset}

With the above described we are able to identify the model for any offset angle φ_{offset}. However, only one value is correct. To find this correct offset value φ^*_{offset} we will perform the identification for multiple different offset angles φ_{offset} and for each of these try to quantify the validity of the identified parameters. We should see a roughly quadratic relationship between the φ_{offset} and the chosen error criterion with the minimum being in φ^*_{offset}.

Using the identified parameters Θ we simulate the model using the input voltage u as the input. As our error criterion we will use the difference between the simulated pitch angle and the pitch angle obtained from the real world experiment. If our mathematical model were to describe the system exactly we would expect and was correctly identified we would expect a error value of 0 over the whole simulation time. Since we are not aiming at a exact model, only at an usable estimate, we are content for the simulation to roughly follow the real world data.

Because of the non-exact nature of our model, its output error will tend to accumulate over the simulation time. Because of this, it would be difficult to simply measure the error value since the simulated dynamics would be different from the real world data. The solution here is to periodically reset the simulation, using the real world data as initial values. This ensures that our error has the same dynamic behaviour over the long term as the real world data while still functioning as a measure of the short-term validity of our model.

With this method the model output over time is

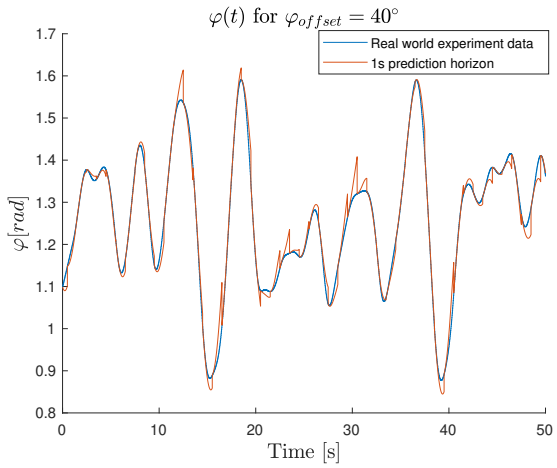


Figure 26: Model simulation with identified parameters where its states are set to real world data every second

This simulation was performed for φ_{offset} between 0 and 100 degrees and for each of them the euclidean norm of error over simulation time was taken.

$$|e| = \sqrt{\sum_{i=0}^{i=N} e_i} \tag{6.36}$$

This norm was then visualized over the whole φ_{offset} space to obtain this figure.

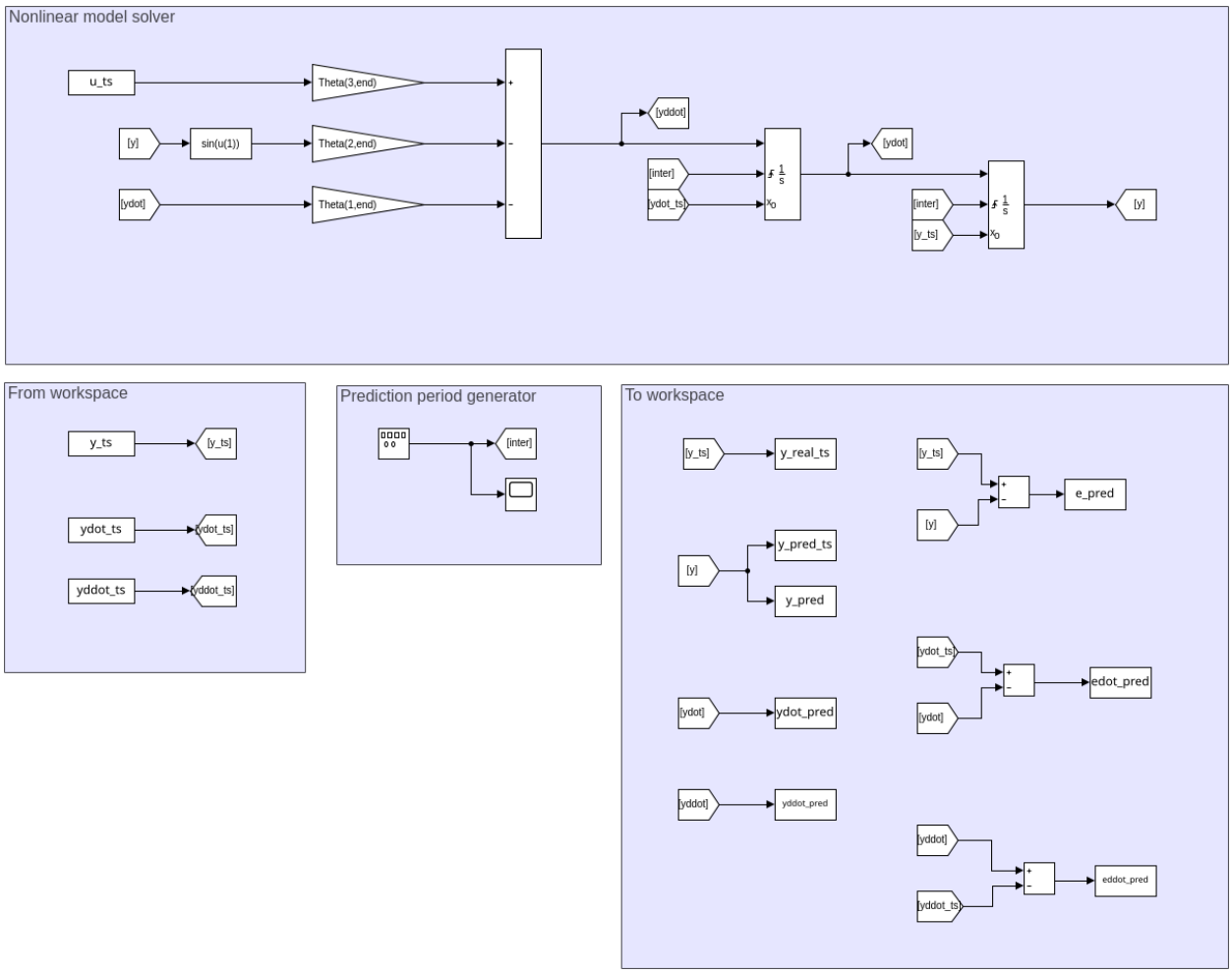


Figure 27: Simulink solution used for error prediction

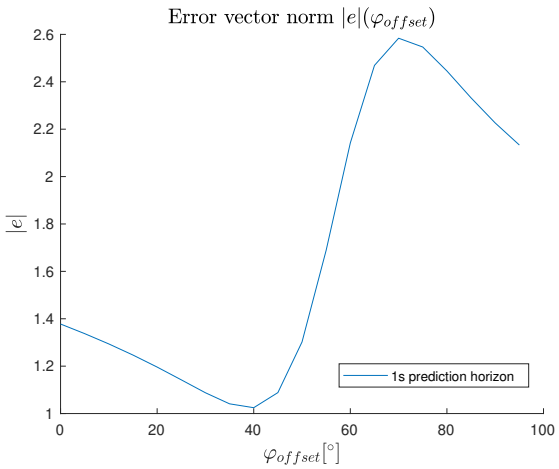


Figure 28: The error of our model compared to experimental data for different φ_{offset}

The model error has a clear local minimum at $\varphi_{offset} = 40°$. This corresponds to an approximate estimate taken when performing the experiment.

Our identified model parameters Θ for $\varphi_{offset} = 40°$ are

$$\Theta = \begin{bmatrix} 0.4554 \\ 3.2503 \\ 13.3097 \end{bmatrix} \tag{6.37}$$

The identified model we will use further for developing control algorithms is

$$\ddot{\varphi} + 0.4554\dot{\varphi} + 3.2503sin(\varphi) = 13.3097u \tag{6.38}$$

The angle sensor data provided by the physical system is always offset by $-40°$ from this mathematical model and has to be manually adjusted to fit correctly. Input $u \in < 0, 1 >$.

6.5.2 Statistical evaluation of identification accuracy

After we have completed the identification process, we can perform a statistical analysis of it to decide whether or not the results are satisfactory.

The analysis we are going to perform is inspired by the theoretical background from [3]. We are going to take the difference between the experiment data and the prediction data measured with the 1s prediction horizon $\mathbf{e}$. We are going to calculate its variance s^2 as

$$s^2 = \frac{\mathbf{e}^T\mathbf{e}}{N} = 1.3e{-}4 \quad \text{where } N \text{ is the length of } \mathbf{e} \tag{6.39}$$

And use it to calculate the covariance matrix $Cov(\Theta)$ of the identified parameters as

$$Cov(\Theta) = s^2(\Phi^T\Phi)^{-1} = 1e^{-3}\begin{bmatrix} 0.0014 & 0.0011 & 0.0046 \\ 0.0011 & 0.0066 & 0.0271 \\ 0.0046 & 0.0271 & 0.1113 \end{bmatrix} \tag{6.40}$$

These values are relatively small compared to the identified parameters, this tells us that we can be relatively certain that the identification was successful. Of course, this method of

statistical evaluation relies on the presupposition that the predictors in Φ are uncorrelated. This is not true, as in our case we compute the velocity and acceleration predictors from the position predictor so these results have to be understood only as rough estimates.

6.5.3 Useful linearized positions

We linearized this identified nonlinear differential equation around 3 points. We will use these linearized systems as a intermediate step in the design process. We will design controllers to control these linearized systems and then we will test whether or not they are able to control the original identified nonlinear system.
The system linearized around $\varphi^* = 90°$ has the following transfer function.

$$F_{90}(s) = \frac{13.31}{s^2 + 0.4554s} \tag{6.41}$$

This system will also function as the internal MPC model used to build the prediction matrices of the MPC control algorithm.

The system linearized around $\varphi^* = 50°$ has the following transfer function.

$$F_{50}(s) = \frac{13.31}{s^2 + 0.4554s + 2.089} \tag{6.42}$$

The system linearized around $\varphi^* = 130°$ has the following transfer function.

$$F_{130}(s) = \frac{13.31}{s^2 + 0.4554s - 2.089} \tag{6.43}$$

7 MPC control

7.1 MPC principle

Model Predictive Control is a control loop mechanism where the controller contains an internal linear model of the controlled plant and for each time step the MPC controller calculates the optimal correction based on predictions calculated over the internal model.

MPC considers the output error e as well as the input u for its optimal control calculations. It can also be expanded to consider the difference Δu between time steps to penalize aggressive changes in control.

Our particular MPC implementation uses definitions described in [4] and [5].

7.1.1 Prediction

The predictions are performed using an internal discrete linear model in the form

$$x_{k+1} = Ax_k + Bu_k \tag{7.1}$$

$$y_k = Cx_k + Du_k \tag{7.2}$$

where $A \in R^{n\times n}$, $B \in R^{n\times 1}$, $C \in R^{1\times n}$, $D \in R^{1\times 1}$, $x_k \in R^{n\times 1}$, $u_k \in R^{1\times 1}$ and n is the system order.

The state vector x_k has to be augmented by the piecewise reference value w in order for the output to track a piecewise signal. It also convenient to augment it the previous input vector $\mathbf{u}$˙-1. The augmented state equation has the following form

$$\underbrace{\begin{bmatrix} x_{k+1} \\ w_{k+1} \\ \Delta u_{k-1} \end{bmatrix}}_{\tilde{x}_{k+1}} = \underbrace{\begin{bmatrix} \mathbf{A} & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_k \\ w_k \\ \mathbf{u}_{-1} \end{bmatrix}}_{\tilde{x}_k} + \underbrace{\begin{bmatrix} \mathbf{B} \\ 0 \\ 0 \end{bmatrix}}_{\tilde{B}} u_k \tag{7.3}$$

$$e_k = \underbrace{\begin{bmatrix} \mathbf{C} & 1 & 0 \end{bmatrix}}_{\tilde{C}_e} \tilde{x}_k \tag{7.4}$$

This formulation considers the output error e as the output of the model. This allows us to directly use it for the optimization.

This state equation is then modified to calculate states further into the future than only one time step. This is done by defining state prediction matrices $\mathbf{P}_x$, $\mathbf{H}_x$ and output prediction matrices $\mathbf{P}$ a $\mathbf{H}$. These allow us to simply compute the system output for known manipulated variable values n_p steps into the future. The scalar n_p is called the prediction horizon length. Increasing this allows us to compute optimal control over longer time frames but at the cost of increased computational complexity.

State prediction matrices:

$$\underbrace{\begin{bmatrix} \tilde{\mathbf{x}}_{k+1} \\ \tilde{\mathbf{x}}_{k+2} \\ \vdots \\ \tilde{\mathbf{x}}_{k+n_p} \end{bmatrix}}_{\tilde{\mathbf{x}}_{k+1,n_p}} = \underbrace{\begin{bmatrix} \tilde{A} \\ \tilde{A}^2 \\ \vdots \\ \tilde{A}^{n_p} \end{bmatrix}}_{\tilde{\mathbf{P}}_x} \tilde{\mathbf{x}}_k + \underbrace{\begin{bmatrix} \tilde{B} & 0 & \cdots \\ \tilde{A}\tilde{B} & \tilde{B} & \cdots \\ \vdots & \vdots & \ddots \\ \tilde{A}^{n_p-1}\tilde{B} & \tilde{A}^{n_p-1}\tilde{B} & \cdots \end{bmatrix}}_{\tilde{\mathbf{H}}_x} \underbrace{\begin{bmatrix} u_k \\ u_{k+1} \\ \vdots \\ u_{k+n_p-1} \end{bmatrix}}_{\mathbf{u}_{k,n_p-1}} \tag{7.5}$$

Output prediction matrices:

$$\underbrace{\begin{bmatrix} \tilde{y}_k \\ \tilde{y}_{k+1} \\ \vdots \\ \tilde{y}_{k+n_p-1} \end{bmatrix}}_{\tilde{\mathbf{y}}_{k,n_p-1}} = \underbrace{\begin{bmatrix} \tilde{C} \\ \tilde{C}\tilde{A} \\ \tilde{C}\tilde{A}^2 \\ \vdots \\ \tilde{C}\tilde{A}^{n_p-1} \end{bmatrix}}_{\tilde{\mathbf{P}}} \tilde{\mathbf{x}}_k + \underbrace{\begin{bmatrix} \tilde{D} & 0 & 0 & \cdots \\ \tilde{C}\tilde{B} & \tilde{D} & 0 & \cdots \\ \tilde{C}\tilde{A}\tilde{B} & \tilde{C}\tilde{B} & \tilde{D} & \cdots \\ \vdots & \vdots & \vdots & \ddots \\ \tilde{C}\tilde{A}^{n_p-2}\tilde{B} & \tilde{C}\tilde{A}^{n_p-3}\tilde{B} & \tilde{C}\tilde{A}^{n_p-4}\tilde{B} & \cdots \end{bmatrix}}_{\tilde{\mathbf{H}}} \underbrace{\begin{bmatrix} u_k \\ u_{k+1} \\ u_{k+2} \\ \vdots \\ u_{k+n_p-1} \end{bmatrix}}_{\mathbf{u}_{k,n_p-1}} \tag{7.6}$$

MPC controller computes the system states n_p steps into the future, but for simpler computation we will only look for optimal control n_c time steps into the future where $n_c \leq n_p$. The manipulated variable is set to the n_c-th value and remains constant. This strategy (called **Move blocking**) front loads the manipulated variable variability, but its not necessarily optimal. Another approach would be to divide the n_c computed u values evenly over the prediction horizon. **Move blocking** is implemented using the following:

$$\tilde{\mathbf{H}}_{b,x} = \tilde{\mathbf{H}}_x \mathbf{M}_b \tag{7.7}$$

$$\tilde{\mathbf{H}}_b = \tilde{\mathbf{H}} \mathbf{M}_b \tag{7.8}$$

Where the matrix $\mathbf{M}_b$ has dimensions $n_p \times n_c$

$$\mathbf{M}_b = \begin{bmatrix} 1 & & \\ & \ddots & \\ & & 1 \\ & & \vdots \\ & & 1 \end{bmatrix} \tag{7.9}$$

We will also define the way the manipulated variable $\mathbf{u}_i$ is transformed into the first order difference $\Delta\mathbf{u}_i$. For each time step the form is $\Delta\mathbf{u}_i = \mathbf{u}_i - \mathbf{u}_{i-1}$. This is expressed in matrix form over the prediction horizon as

$$\underbrace{\begin{bmatrix} \Delta\mathbf{u}_k \\ \Delta\mathbf{u}_{k+1} \\ \vdots \\ \Delta\mathbf{u}_{k+n_c-1} \end{bmatrix}}_{\Delta\mathbf{u}_{k,n_c-1}} = \underbrace{\begin{bmatrix} \mathbf{I} & 0 & 0 & \cdots \\ -\mathbf{I} & \mathbf{I} & 0 & \cdots \\ 0 & -\mathbf{I} & \mathbf{I} & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{bmatrix}}_{\mathbf{K}\in R^{n_c \times n_c}} \underbrace{\begin{bmatrix} \mathbf{u}_k \\ \mathbf{u}_{k+1} \\ \vdots \\ \mathbf{u}_{k+n_c-1} \end{bmatrix}}_{\mathbf{u}_{k,n_c-1}} + \underbrace{\begin{bmatrix} -\mathbf{I} \\ 0 \\ \vdots \\ 0 \end{bmatrix}}_{\mathbf{M}} \mathbf{u}_{k-1} \tag{7.10}$$

The last term $\mathbf{u}_{k-1}$ can be obtained from our augmented state vector $\tilde{\mathbf{x}}_k$ as

$$\mathbf{u}_{k-1} = \underbrace{\begin{bmatrix} 0 & 0 & \mathbf{I} \end{bmatrix}}_{\tilde{L}} \tilde{x}_k \tag{7.11}$$

With this we have defined how to compute the behaviour or the controlled plant over the prediction horizon. Next we will use this information to compute the optimal control, to minimize a selected cost function.

7.1.2 Optimal control

To use the term optimal, we have to decide on a cost function that will quantify our loop performance. For our purposes we will use the following *cost function*

$$J = \frac{1}{2}(\sum_{i=k}^{k+n_p-1} \mathbf{e}_i^T \mathbf{Q} \mathbf{e}_i + \sum_{i=k}^{k+n_c-1} \Delta \mathbf{u}_i^T \mathbf{R} \Delta \mathbf{u}_i) \tag{7.12}$$

This cost function weighs the deviation of our plant from reference e and the manipulated variable first order differences Δu. These are multiplied by the weighting matrices $\mathbf{Q} \in R^{n_p \times n_p}$ and $\mathbf{R} \in R^{n_c \times n_c}$. These weighting matrices allow us to better customize our cost function. $\mathbf{Q}$ and $\mathbf{R}$ have to be symmetric and positive-definite. The lower the value of J, the closer we are to our desired state. We will therefore use this function to compute the optimal u.

This cost function can be expressed using our above defined prediction matrices using algebraic manipulations as

$$J = \frac{1}{2}(\tilde{\mathbf{P}} \tilde{\mathbf{x}}_k + \tilde{\mathbf{H}}_b \mathbf{u}_{k,n_c-1})^T \mathbf{Q} (\tilde{\mathbf{P}} \tilde{\mathbf{x}}_k + \tilde{\mathbf{H}}_b \mathbf{u}_{k,n_c-1}) \tag{7.13}$$

$$+\frac{1}{2}(\mathbf{K}\mathbf{u}_{k,n_c-1} + \mathbf{M}\mathbf{L}\tilde{\mathbf{x}}_k)^T \mathbf{R} (\mathbf{K}\mathbf{u}_{k,n_c-1} + \mathbf{M}\mathbf{L}\tilde{\mathbf{x}}_k) \tag{7.14}$$

This is then expanded and simplified to obtain the following quadratic function

$$J = \frac{1}{2}\mathbf{u}_{k,n_c-1}^T \mathbf{G} \mathbf{u}_{k,n_c-1} + \mathbf{f}^T \mathbf{u}_{k,n_c-1} + c, \tag{7.15}$$

where

$$\mathbf{G} = \tilde{\mathbf{H}}_b^T \mathbf{Q} \tilde{\mathbf{H}}_b + \mathbf{K}^T \mathbf{R} \mathbf{K} \tag{7.16}$$

$$\mathbf{f}^T = \tilde{\mathbf{x}}_k^T (\tilde{\mathbf{P}}^T \mathbf{Q} \tilde{\mathbf{H}}_b + \mathbf{L}^T \mathbf{M}^T \mathbf{R} \mathbf{K}) \tag{7.17}$$

$$c = \frac{1}{2}\tilde{\mathbf{x}}_k^T (\tilde{\mathbf{P}}^T \mathbf{Q} \tilde{\mathbf{P}} + \mathbf{L}^T \mathbf{M}^T \mathbf{R} \mathbf{L} \mathbf{M}) \tilde{\mathbf{x}}_k \tag{7.18}$$

The optimal control is then computed by solving the following quadratic optimization problem for each time simulation time step

$$\text{minimize } \frac{1}{2}\mathbf{u}_{k,n_c-1}^T \mathbf{G} \mathbf{u}_{k,n_c-1} + \mathbf{f}^T \mathbf{u}_{k,n_c-1} + c \tag{7.19}$$

$$\text{over } \mathbf{u}_{k,n_c-1} \text{ with subject to constraints} \tag{7.20}$$

This quadratic programming problem is solved at each time step k to compute the control law $\mathbf{u}_{k,n_c-1}$. The first step of this vector is applied to the controlled plant as the MPC manipulated variable action. The rest is used for future prediction that is used to form the cost function at the next time step $k+1$.

7.1.3 Unconstrained MPC

When no constraints are present in a MPC controller, the controller reduces to a time-invariant state feedback controller as shown in [4].
The QP optimality condition looks for the minimum of J, meaning it searches for $gradJ = 0$. The gradient of the objective function is $\mathbf{Gu} + \mathbf{f}$. This can be simply solved as $\mathbf{u} = -\mathbf{G}^{-1}\mathbf{f}$. Since $\mathbf{u}$ is a vector in $\mathrm{R}^{n_c \times 1}$, we need to select just the first input to be applied. This is done by

$$u_k = -\mathbf{N}\mathbf{G}^{-1}\mathbf{f} \tag{7.21}$$

$$, \text{where} \quad \mathbf{N} = \begin{bmatrix} 1 & 0 & \dots & 0 \end{bmatrix} \in \mathrm{R}^{1 \times n_c}$$

We can substitute $\mathbf{G}$ and $\mathbf{f}$ using the equations (7.16) and (7.17) to obtain u_k in the following form

$$u_k = -\underbrace{\mathbf{N}(\tilde{\mathbf{H}}_b^T \mathbf{Q} \tilde{\mathbf{H}}_b + \mathbf{K}^T \mathbf{R} \mathbf{K})^{-1}(\tilde{\mathbf{H}}_b^T \mathbf{Q} \tilde{\mathbf{P}} + \mathbf{K}^T \mathbf{R} \mathbf{M} \mathbf{L})}_{\mathbf{Z}} \hat{\mathbf{x}}_k \tag{7.22}$$

The matrix $\mathbf{Z}$ can be divided into three submatrices, corresponding to the original physical states, the augmented reference value and the last controller output u_{-1}.

$$u_k = -\begin{bmatrix} \mathbf{Z}_1 & \mathbf{Z}_2 & \mathbf{Z}_3 \end{bmatrix} \begin{bmatrix} x_k \\ w_k \\ u_{-1} \end{bmatrix} \tag{7.23}$$

During this analysis we will assume that we are able to directly measure the states of the controlled plant x_k. This is equivalent to setting its output matrix $\mathbf{C}$ to an identity matrix. In reality, we would use a state estimator, but that would complicate the analysis process, as the estimator would introduce additional dynamic behaviour into the loop.
The transfer function F_C is the controlled plant's transfer function with the output matrix set to an identity matrix. The schema implementing the equation (7.23) is

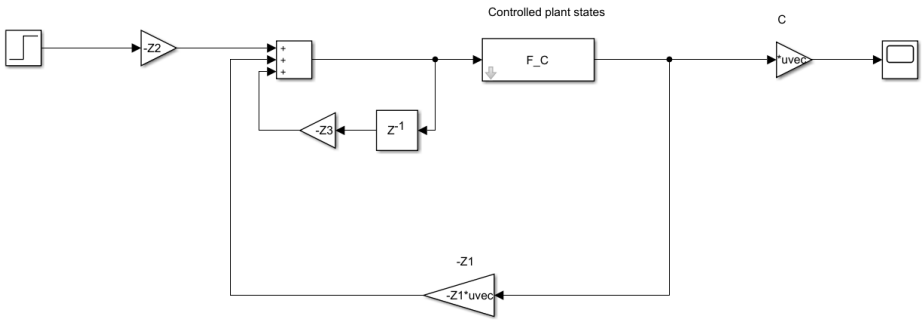


Figure 29

Using simple block algebra, we separate the part computing u_{-1} into a block in series with the controlled plant $F_C(z)$

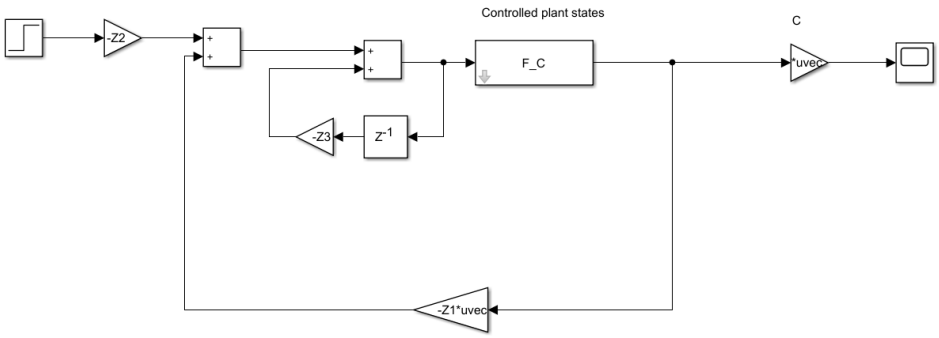


Figure 30

The separated block is described by this difference equation

$$u_{k+1} = -\mathbf{Z}_3 u_k + u'_k \tag{7.24}$$

where u_k is the output and u'_k is the input. We transform this equation into a transfer function of z using the Z-transform.

$$zU = -\mathbf{Z}_3 U + U' \tag{7.25}$$

$$F_u(z) = \frac{U}{U'} = \frac{z}{z + \mathbf{Z}_3} \tag{7.26}$$

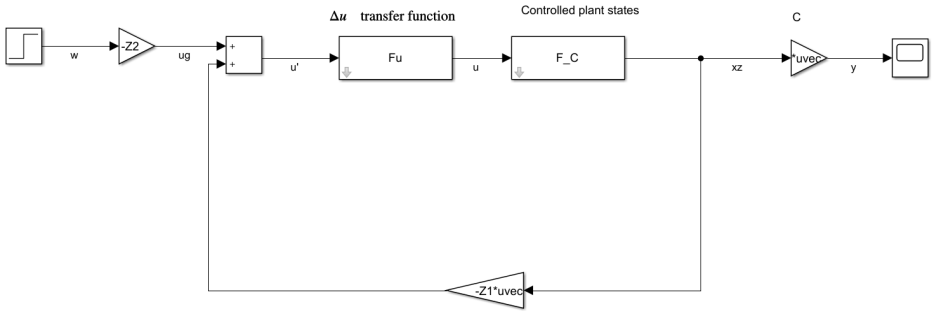


Figure 31

We define $F(z) = F_u(z) \cdot F_C(z)$ as the series connection of the two transfer functions. The transfer function G from u_g to x_z can be then infered using the following equations made using the schematic in 31. During this derivation we have to be mindful of the fact that $\mathbf{Z}_1 \in \mathrm{R}^{n\times 1}$ and the output of F_C is a $n \times 1$ vector, where n is the system order. Meaning we are working with vectors of transfer functions and have to rely on matrix properties when doing algebra.

$$X_z(z) = F \cdot (U_g(z)\mathbf{Z}_1 X_z(z)) \tag{7.27}$$
$$(\mathbf{I} + F(z)\mathbf{Z}_1)X_z(z) = F(z)U_g(z) \tag{7.28}$$
$$X_z(z) = (\mathbf{I} + F(z)\mathbf{Z}_1)^{-1}F(z)U_g(z) \tag{7.29}$$
$$G(z) = (\mathbf{I} + F(z)\mathbf{Z}_1)^{-1}F(z) \tag{7.30}$$

$G(z)$ is a $n \times 1$ vector of transfer functions. The scalar feedback loop transfer function (w to y) $F_z(z)$ is then derived as

$$F_z(z) = -\mathbf{C}G(z)\mathbf{Z}_2 \tag{7.31}$$

Another way to interpret this is that the u_{-1} introduces an additional state r_k into the closed loop, where

$$x_{k+1} = Ax_k + Br_k \tag{7.32}$$
$$r_{k+1} = -Z_3 r_k - Z_1 x_k - Z_2 w_k \tag{7.33}$$

Making the closed loop dynamics more complicated than is usually the case for a classical LQR controller.

Yet another useful finding from this schema is the open loop transfer function from the controlled plant input to the controller output

$$F_o(z) = -F_u(z)\mathbf{Z}_1 F_C(z) \tag{7.34}$$

This transfer function is useful because it allows us to find the stability margins of the control loop.

Unconstrained MPC can be used to gain an insight into the control loop stability, as it allows us to see the closed loop poles and therefore to use classical frequency domain methods. The stability of a constrained MPC control loop is however much more difficult to theoretically prove. This is because the constraints cause the MPC controller to behave in a nonlinear fashion. Therefore, the stability of a MPC control loop is most often investigated using simulations.

7.2 State estimator for input disturbance rejection

The real world model is under influence of a number of unmodeled behaviours. The effects of gravity change as the model moves farther from the angle around which the controller was designed. The motor used to move the helicopter also has an influence on the dynamics that are not modeled. Totally unknown phenomena can also influence it, such as wind.

All of these can be approximately modeled as an input disturbance acting on our physical model. We make the assumption that these disturbances have dynamics close to constant and can therefore be generally modelled using the following state space model.

$$G_z : \dot{x_z}(t) = 0; \qquad x_z(t_0) = 1 \tag{7.35}$$

$$z(t) = x_z(t) \tag{7.36}$$

This model is a first order integrator system, whose pole lies in 0.
The output of this system $z(t)$ is not directly measurable, but we can build an identical estimator to estimate it.
The discrete linearized model used to build the MPC controller serves as a baseline for this estimator. The state matrices (A,B,C,D) were augmented by an additional integrator state. This is because we assume the input disturbance to be a piecewise function whose internal model is an integrator. Therefore we include an additional integrator state into the estimator. This state will estimate the current input disturbance present in the system allowing us to use this estimation to compensate the presence of the disturbance.
Since we are designing the estimator directly in the Z-domain, the integrator is introduced as a state with its pole in 1.

$$A_{est} = \begin{bmatrix} A & B & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{7.37}$$

$$B_{est} = \begin{bmatrix} B \\ 0 \end{bmatrix} \tag{7.38}$$

$$C_{est} = \begin{bmatrix} C & 0 \end{bmatrix} \tag{7.39}$$

When designing this estimator we need to chose its poles. We need to make sure the estimator is stable, but beyond that we need to select a viable compromise between speed of estimation and noise amplification. Placing all the estimator's poles in 0 would result in a deadbeat response, which has the fastest speed of estimation, but greatly amplifies any noise present. This is not a viable solution. Instead, we chose to place the poles in $\begin{bmatrix} 0.4 & 0.4 & 0.4 \end{bmatrix}$ making the estimation speed slower, but ensuring any noise does not get unreasonably amplified. We used the Matlab command *acker()* to obtain the innovation matrix L_c.
This innovation matrix was then used to obtain the estimator's matrix of dynamics $F_{est} = A_{est} - L_c C_{est}$.

This estimator is capable of estimating the input disturbance present in the system. We can then use this information to subtract the compensate for the disturbance by subtracting it from the MPC control output. This essentially eliminates the effects of the input disturbance, assuming our estimation is correct. In the physical system, this subtracting would be done by the computer running the MPC algorithm. Meaning we need to make sure we do not

overstep the bound the MPC control loop operates in. To do this, we add the estimated disturbance to the MPC constraints lb_u and ub_u in each MPC time step.

7.3 MPC constraints for helicopter model

We will discuss the use of Model Predictive Control. We will use a linearized model of our system in a equilibrium point as our internal model because of the ease of calculating the predictions.

To solve the quadratic programming problem in our simulation we use the Matlab command *quadprog()* (Later also qpOASES). This allows us to easily set constraints of our system. Most important of these is the limitation on the manipulated variable values. The pitch actuator of our helicopter model operates on values between 0 (no torque) to 1 (maximum torque). Any control outside this range is nonsensical for our system. We therefore define the following bounding vectors

$$\mathbf{ub}_u = \underbrace{\begin{bmatrix} 1 \\ 1 \\ \vdots \end{bmatrix}}_{\in \mathrm{R}^{n_c \times 1}} \qquad \mathbf{lb}_u = \underbrace{\begin{bmatrix} 0 \\ 0 \\ \vdots \end{bmatrix}}_{\in \mathrm{R}^{n_c \times 1}} \tag{7.40}$$

The bounded quadratic programming problem then becomes

$$\text{minimize } \frac{1}{2}\mathbf{u}_{k,n_c-1}^T \mathbf{G}\mathbf{u}_{k,n_c-1} + \mathbf{f}^T \mathbf{u}_{k,n_c-1} + c \tag{7.41}$$

$$\text{over } \mathbf{u}_{k,n_c-1} \tag{7.42}$$

$$\text{where } \mathbf{lb}_u \leq \mathbf{u}_{k,n_c-1} \leq \mathbf{ub}_u \tag{7.43}$$

7.4 Showcase of MPC control

The MPC controller described above was tested on both the linearized model and the nonlinear model. The controller has to be able to control the nonlinear model, since that's the closest mathematical description we have of the real-world system.

These simulations were performed with the following parameters

n_p	50
n_c	10
$\mathbf{Q}$	$\mathbf{I}$
$\mathbf{R}$	$0.1\mathbf{I}$

Table 1

The ratio between $\mathbf{Q}$ and $\mathbf{R}$ determines the aggressiveness of the controller. Higher $\mathbf{R}$ forces the controller to use gentler action, which we can use to avoid exciting the unmodeled nonlinear behaviour of the real-world model at the cost of more overshoot and slower time response.

Linear plant model

In this simulation we set the controlled system to be the linear version of the equation of motion in equation (6.38), linearized around φ^*.

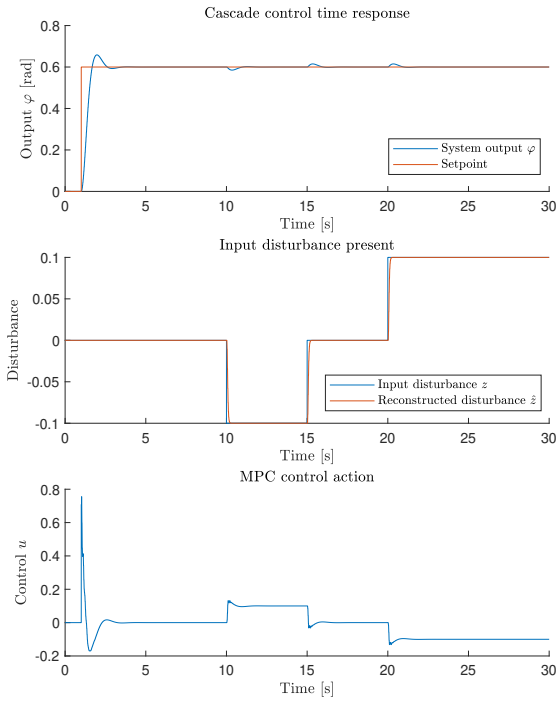


Figure 32: MPC control for system linearized around $\varphi^* = 90°$

The MPC controller uses an internal model of the system. This internal model is identical to the system linearized around $\varphi^* = 90°$. We can use a different linear model for the internal model, but this one is the closest to the state we will want to use the helicopter model in. As we can see, the controller is able to follow the reference signal and is also able to reject the input disturbance z.

Nonlinear plant model

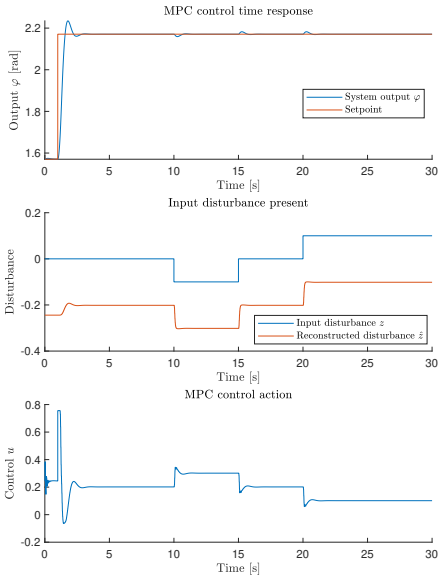
For this simulation, the MPC controller is used to control the nonlinear model of our helicopter system. That means that the internal model in the controller is not an exact match to the controlled plant. They should match when close to the angle $\varphi^* = 90° = 1.57rad$. But the farther from this point we move, the less the MPC controller understands the controlled plant.

Since the MPC controller uses a linear version of the controlled plant that is linearized around 90° we need to make some adjustments to use it to control the nonlinear model.

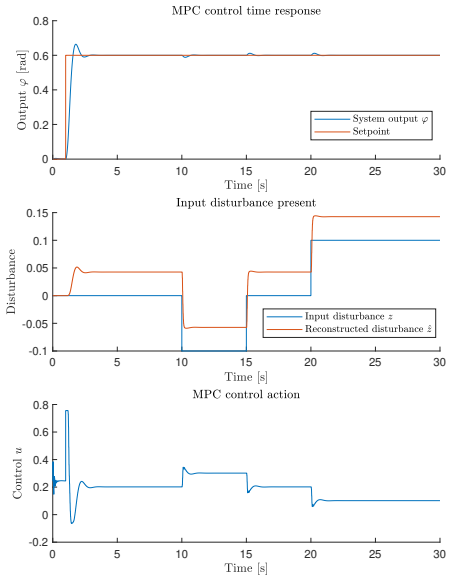
- The angle φ_{NS} measured as the nonlinear system output is the physical angle present. However, since the MPC controller uses an internal model linearized around φ^*, we need to subtract φ^* from the measured nonlinear system output before we use it to estimate the system states. $\varphi_{MPC} = \varphi_{NS} - \varphi^*$
- Similarly, the reference value $w = 0$ to the MPC controller corresponds to the nonlinear

system output 90°. Meaning we need to add φ^* to the reference value for it to correctly line up with the nonlinear system output φ_{NS}.

- The linear model assumes a steady-state input u_0 to hold the nonlinear system steady in φ^*. When starting the simulation with the nonlinear system initial conditions $\varphi_{NS} = \varphi^*, \dot{\varphi_{NS}} = 0$ the output would immediately start falling until the estimator managed to estimate $\hat{z} = -u_0$ (Negative, because $u(k) = u_{MPC}(k) - \hat{z}(k)$, see (7.2)), which would hold the system in steady state. We can set the estimator's initial conditions to $\begin{bmatrix} 0 & 0 & -u_0 \end{bmatrix}$ to prevent this.



(a) True values

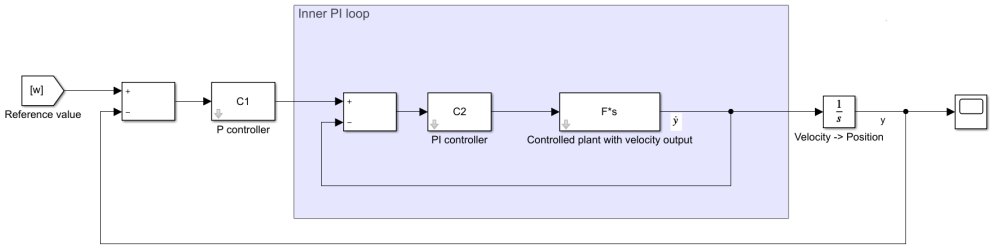


(b) Values from the point of view of the MPC controller

The MPC controller is able to reliably control the nonlinear system. The output reaches the setpoint and the controller rejects disturbances soon after they appear. In the figure 33b, the estimated disturbance $\hat{z}$ is rid of its u_0 part. Even then the estimated disturbance doesn't match the disturbance we apply directly. This is because as the nonlinear model gets farther from the internal model of the MPC controller is less and less accurate. This inaccuracy is modeled as the additional input disturbance estimated and allows the loop to reach steady state even when the internal MPC model does not match the controlled plant exactly.

7.5 Comparison to classical cascade control

We will use a cascade control scheme as a baseline for evaluating the performance of our MPC control law.



The plant F's velocity (Our model plant augmented by adding a zero to its transfer function) is controlled by a PI controller $C_2(s)$. This control loop's is then integrated to obtain the position which is controlled by another outer feedback loop using a P controller $C_1(s)$.

We designed the inner PI controller using MATLAB Control System Designer as:

$$C_2(s) = \frac{0.387s + 0.2528}{s} \tag{7.44}$$

and the outer P controller

$$C_1(s) = 1.3355 \tag{7.45}$$

This designed cascade loop was tested on 3 linearized positions of the nonlinear helicopter model, namely (6.42), (6.41), (6.43). Figure 33 shows the time responses of this simulation.

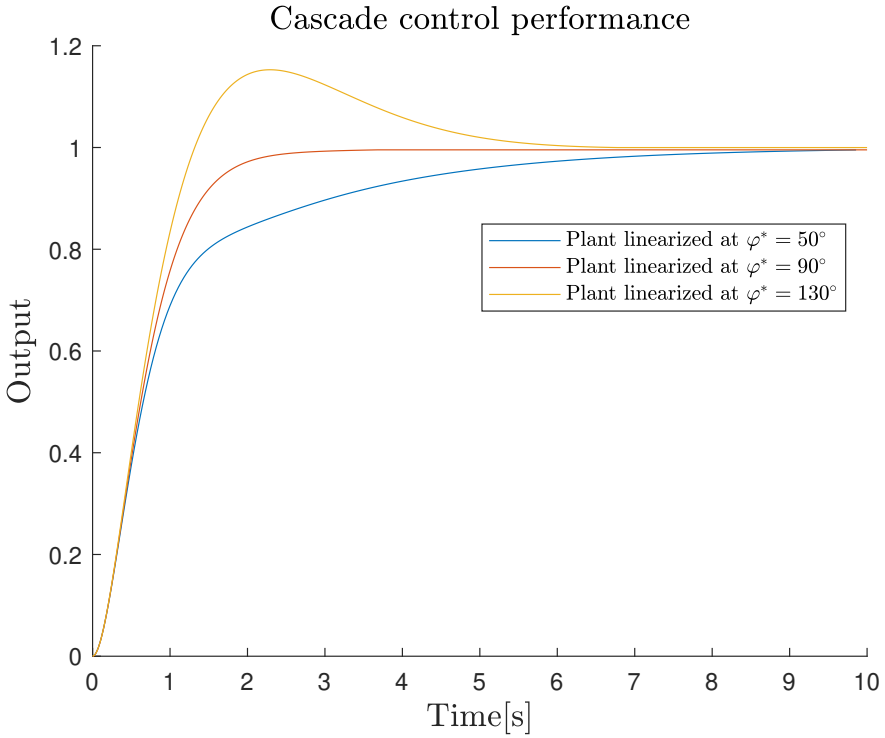


Figure 33: Cascade control scheme used for control of the nonlinear plant linearized at various equilibrium points

The control loop is reasonably robust, being able to bring the 50° and 130° systems to reference.

Input disturbances are present in our real-world model. The disturbances enter our control loop as can be seen in figure 34

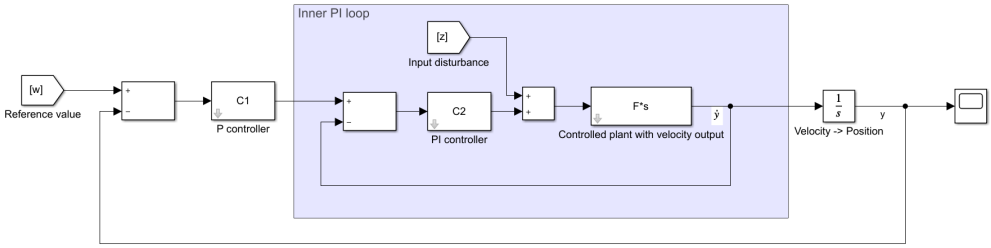


Figure 34: Input disturbances present in our cascade loop

The cascade loop has an intrinsic ability to reject these input disturbances, as can be seen from figure 35

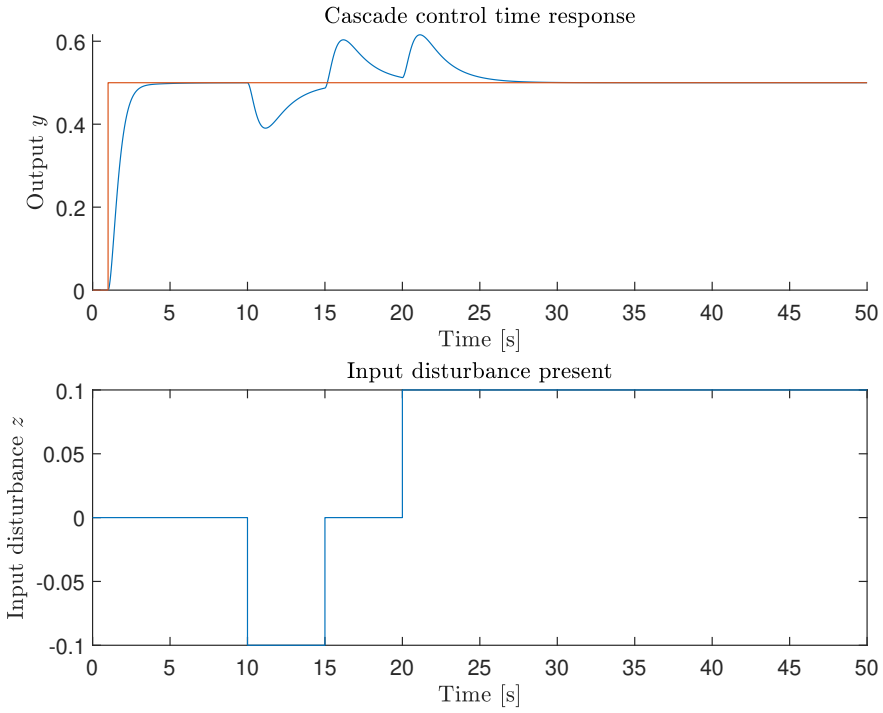


Figure 35: Disturbance rejection capability

To bring this control loop more in line with the physical reality, we need to discretize the controllers. We will discretize them with the zero-order hold method, with sample time $T_s = 0.02s$ to match that of the MPC controller. The discretized controllers become

$$C_{2d}(z) = \frac{0.387(z - 0.9869)}{z - 1} \tag{7.46}$$

$$C_{1d}(z) = 1.3355 \tag{7.47}$$

Since we are taking a derivative of the position by the zero in origin introduced for velocity control, it is useful to add a low-pass filter after the derivative to prevent any high-frequency noise amplification. We found out earlier that all useful dynamics or our system are below $1Hz$. We will therefore use a simple low-pass single order discrete filter with passband frequency $1Hz$ in the form of

$$G(z) = \frac{\frac{T_s}{T} z^{-1}}{1 + \frac{T_s}{T-1} z^{-1}} \tag{7.48}$$

where $T_s = 0.02$ is the sample time and $T = \frac{1}{2\pi 1}$ is the filter time constant that corresponds to the $1Hz$ passband.

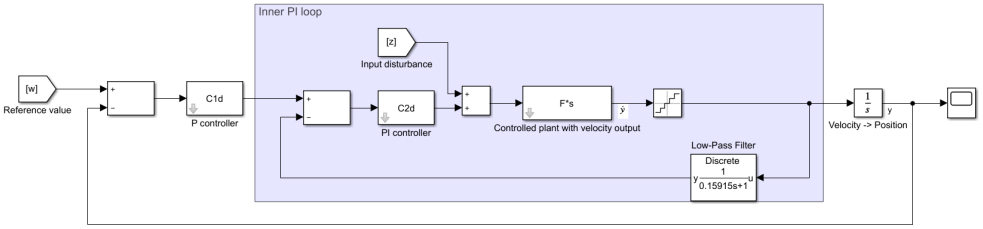


Figure 36: Cascade control loop with velocity filtration

In our real-world model, the only sensor available measures the angle and has a limited quantization step of roughly $0.00628rad$. To get an approximation of the velocity we will need to take the numerical difference from the position data. The position data will be weighted down by the quantization. For simplicity, we will use the first order backward difference defined as

$$\dot{y}_{num}(k) = \frac{y(k) - y(k-1)}{T_s} \tag{7.49}$$

This numerical differentiation process is an implementation of the zero added to the transfer function.

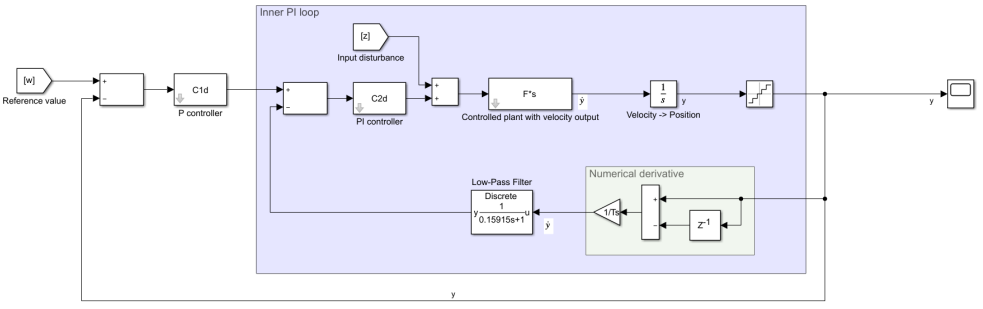


Figure 37: Cascade control loop with quantization and numerical differentiation

Figure 38 shows how the capability to reject disturbances slightly worsens when the schematic includes a quantizer. This is to be expected, as the schematic without the quantizer is more idealized and less in line with our physical system.

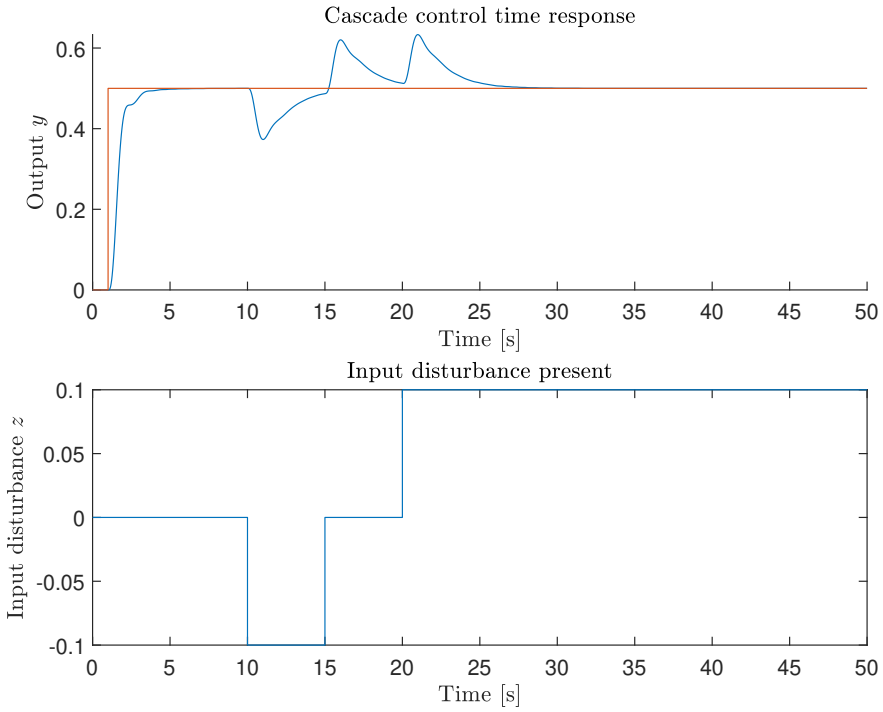


Figure 38: Disturbance rejection capability with filtration and quantization

7.5.1 Simulation results

The performance of the cascade control loop was compared with the performance of the MPC control loop. We simulated 5 different scenarios and measured the process variable and the manipulated variable time response for both control schemes. The different scenarios are:

1. Different reference step amplitudes. Input disturbance present. Controlled plant was designed for.
2. Different reference step amplitudes. Input disturbance present. Controlled plant is a different linear system.
3. Different reference step amplitudes. Input disturbance present. Controlled plant is a different linear system.
4. Different reference step amplitudes. Input disturbance present. Controlled plant is a model of the nonlinear system.

For each time response we evaluated its ITAE value defined as

$$ITAE = \int_0^{t_{end}} ty(t)dt \tag{7.50}$$

and the system input energy defined as

$$E_u = \int_0^{t_{end}} u(t)^2 dt \tag{7.51}$$

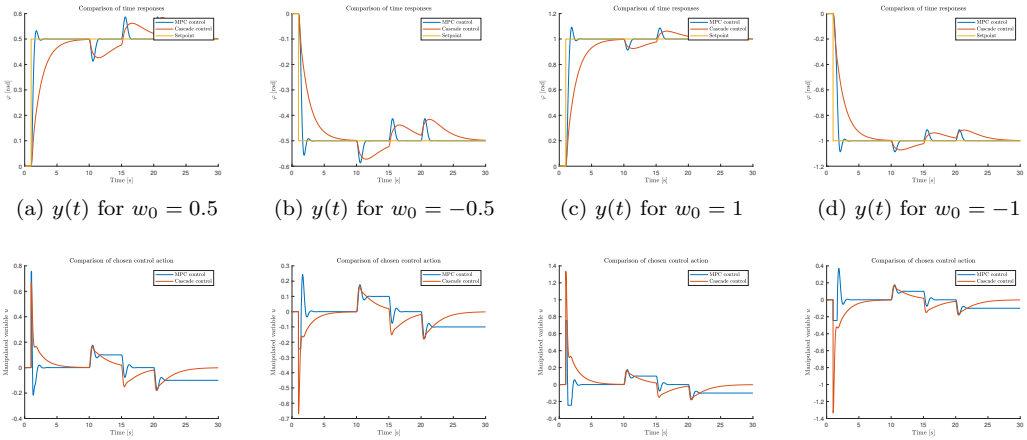
The MPC controller used $0.02s$ sample time, $n_p = 50, n_c = 10$. The PID controllers were discretised to the same sample time.

The MPC control input is bounded using hard constraints. This is because of the physical limitations of our real-world system. This means it struggles more as the reference values increase in amplitude. For this comparison we did not consider the cascade input bounded the same way, as that would introduce a nonlinear element into the cascade control loop.

Scenario 1 The control loops were simulated using different reference step amplitudes (0.5, -0.5, 1, -1). Input disturbance in the form of

$$z(t) = \begin{cases} 0 & t \leq 10 \\ -0.1 & 10 \leq t \leq 15 \\ 0 & 15 \leq t \leq 20 \\ 0.1 & 20 \leq t \end{cases} \tag{7.52}$$

Both control loops were designed around the nonlinear helicopter model linearized around $\varphi^* = 90°$.



(a) $y(t)$ for $w_0 = 0.5$ (b) $y(t)$ for $w_0 = -0.5$ (c) $y(t)$ for $w_0 = 1$ (d) $y(t)$ for $w_0 = -1$

(e) $u(t)$ for $w_0 = 0.5$ (f) $u(t)$ for $w_0 = -0.5$ (g) $u(t)$ for $w_0 = 1$ (h) $u(t)$ for $w_0 = -1$

Figure 39: Time response comparison for exact system model

	ITAE		E_u	
	MPC	**Cascade**	**MPC**	**Cascade**
$w_0 = 0.5$	462	2452	41	37
$w_0 = -0.5$	477	2445	30	37
$w_0 = 1$	516	2738	54	103
$w_0 = -1$	561	2724	36	103

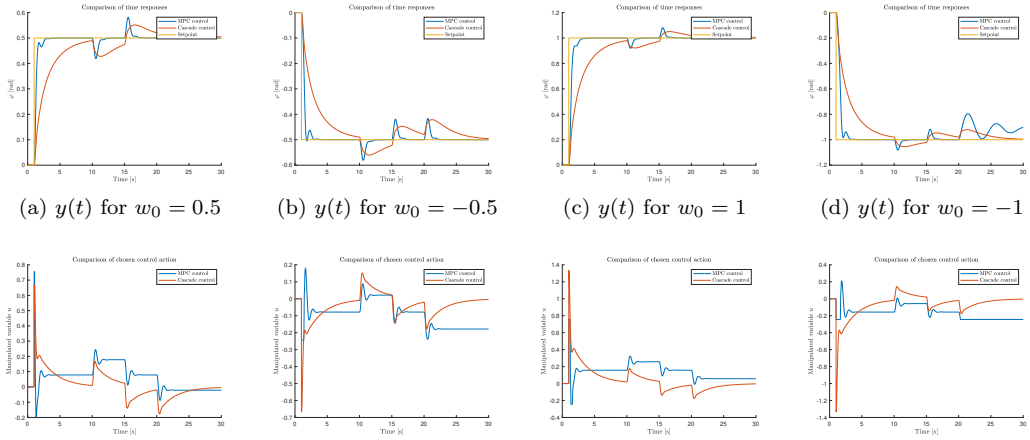
Both control schemes have comparable amount of overshoot when rejecting disturbances, however, the time is takes for the cascade control to reject this disturbance is significantly longer.

Scenario 2 The control loops were simulated using different reference step amplitudes (0.5, -0.5, 1, -1). Input disturbance in the form of

$$z(t) = \begin{cases} 0 & t \leq 10 \\ -0.1 & 10 \leq t \leq 15 \\ 0 & 15 \leq t \leq 20 \\ 0.1 & 20 \leq t \end{cases} \tag{7.53}$$

Both the MPC loop and the cascade loop use their own methods of disturbance rejection.

Both control loops were designed around the nonlinear helicopter model linearized around $\varphi^* = 90°$ (transfer function in (6.41)). For this scenario we set the controlled plant to the same nonlinear helicopter mode, but linearized around $\varphi^* = 50°$ (transfer function in (6.42)). This means that the controlled plant does not match the plant the controllers were designed for, meaning this scenario is an useful indicator of the robustness of the controllers.



(a) $y(t)$ for $w_0 = 0.5$ (b) $y(t)$ for $w_0 = -0.5$ (c) $y(t)$ for $w_0 = 1$ (d) $y(t)$ for $w_0 = -1$

(e) $u(t)$ for $w_0 = 0.5$ (f) $u(t)$ for $w_0 = -0.5$ (g) $u(t)$ for $w_0 = 1$ (h) $u(t)$ for $w_0 = -1$

Figure 40: Time response comparison for plant linearized around $\varphi^* = 50°$

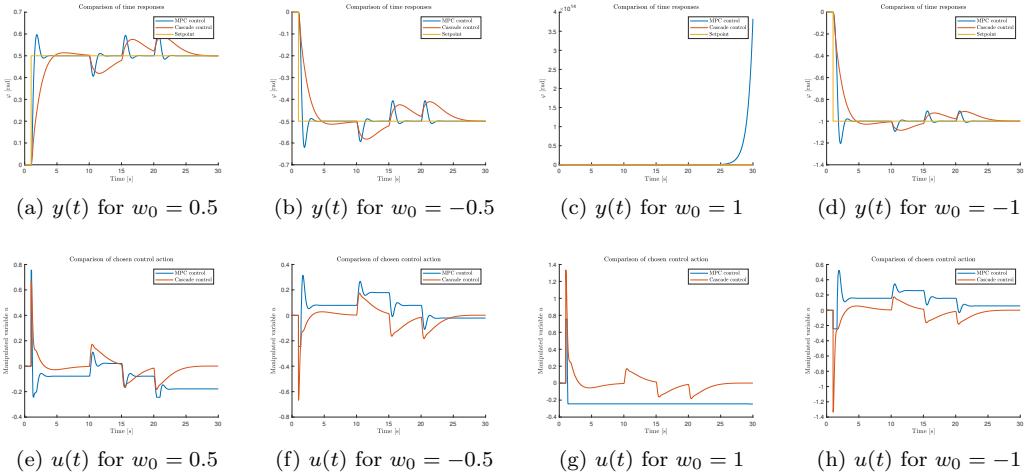
	$ITAE$		E_u	
	MPC	**Cascade**	**MPC**	**Cascade**
$w_0 = 0.5$	469	25595	54	41
$w_0 = -0.5$	487	24998	66	41
$w_0 = 1$	529	30347	131	126
$w_0 = -1$	3671	29179	147	124

The most important result of this scenario is that in the case of $w_m = -1$, the MPC controller was not able to reach the setpoint. This is caused by the MPC being constrained and not being able to produce negative enough outputs.

Scenario 3 The control loops were simulated using different reference step amplitudes (0.5, -0.5, 1, -1). Input disturbance in the form of

$$z(t) = \begin{cases} 0 & t \leq 10 \\ -0.1 & 10 \leq t \leq 15 \\ 0 & 15 \leq t \leq 20 \\ 0.1 & 20 \leq t \end{cases} \tag{7.54}$$

Both control loops were designed around the nonlinear helicopter model linearized around $\varphi^* = 90°$ (transfer function in (6.41)). For this scenario we set the controlled plant to the same nonlinear helicopter mode, but linearized around $\varphi^* = 130°$ (transfer function in (6.43)). This means that the controlled plant does not match the plant the controllers were designed for, meaning this scenario is an useful indicator of the robustness of the controllers.



(a) $y(t)$ for $w_0 = 0.5$ (b) $y(t)$ for $w_0 = -0.5$ (c) $y(t)$ for $w_0 = 1$ (d) $y(t)$ for $w_0 = -1$

(e) $u(t)$ for $w_0 = 0.5$ (f) $u(t)$ for $w_0 = -0.5$ (g) $u(t)$ for $w_0 = 1$ (h) $u(t)$ for $w_0 = -1$

Figure 41: Time response comparison for plant linearized around $\varphi^* = 130°$

The characteristic numbers for the time responses are

	$ITAE$		E_u	
	MPC	**Cascade**	**MPC**	**Cascade**
$w_0 = 0.5$	626	2459	83	37
$w_0 = -0.5$	591	1470	48	37
$w_0 = 1$	+inf	2669	282	94
$w_0 = -1$	688	2690	127	94

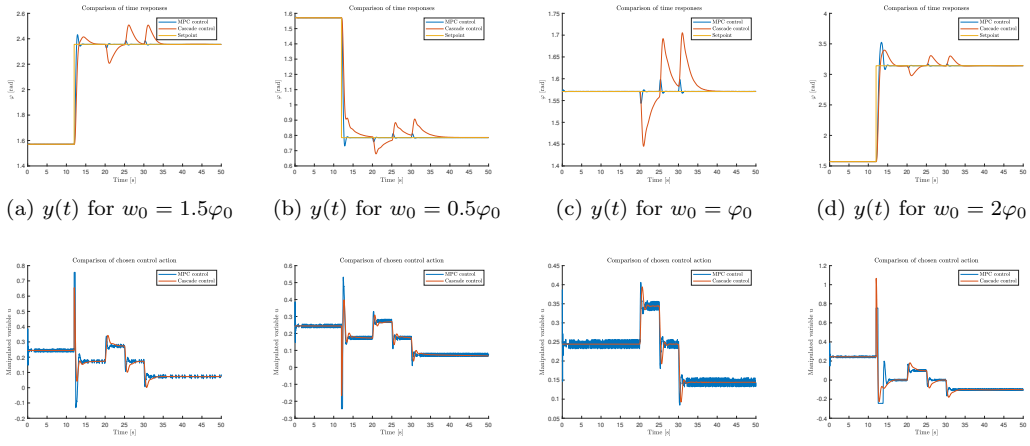
The most important finding of this scenario is that the system goes unstable for $w_0 = 1$ in the case of the MPC controller, while the cascade loop remains stable. This is a potentially dangerous situation. Otherwise, both control schemes retain much of their performance.

Scenario 4 This simulation was performed on the nonlinear system model. Input disturbance in the form of

$$z(t) = \begin{cases} 0 & t \leq 10 \\ -0.1 & 10 \leq t \leq 15 \\ 0 & 15 \leq t \leq 20 \\ 0.1 & 20 \leq t \end{cases} \tag{7.55}$$

Both the MPC loop and the cascade loop used its own estimators for disturbance rejection. Both control loops were designed around the nonlinear helicopter model linearized around $\varphi^* = 90°$. However, they were used to control the nonlinear helicopter model identified earlier. Also, the Simulink block Quantizer was used with 0.0063 quantization step, corresponding to rougly $\frac{1}{3}$ of a degree estimated on the physical optical sensor present on the helicopter model. This quantization introduces a degree of measurement noise into our system that will also be present in the real world model and allows us to gauge how the control schemes will behave under such conditions. The simulations were performed with the nonlinear system having the following initial conditions: $\varphi_0 = \frac{\pi}{2}, \dot{\varphi}_0 = 0$.

For these simulations we slowed the speed of the estimator to $\begin{bmatrix} 0.7 & 0.7 & 0.7 \end{bmatrix}$.This is because the estimator also acts as a lowpass filter with slower estimation speeds resulting in lower passband frequencies. This is an useful property as it allows us to filter some of the measurement noise. Too slow of a estimation causes the estimator dynamics to negatively affect the control loop, but on the other hand, fast estimation increases the measurement noise present. The chosen estimator poles are a good compromise between these two facts.



(a) $y(t)$ for $w_0 = 1.5\varphi_0$ (b) $y(t)$ for $w_0 = 0.5\varphi_0$ (c) $y(t)$ for $w_0 = \varphi_0$ (d) $y(t)$ for $w_0 = 2\varphi_0$

(e) $u(t)$ for $w_0 = 1.5\varphi_0$ (f) $u(t)$ for $w_0 = 0.5\varphi_0$ (g) $u(t)$ for $w_0 = \varphi_0$ (h) $u(t)$ for $w_0 = 2\varphi_0$

Figure 42: Time response comparison when controlling the nonlinear helicopter model

The characteristic numbers for the time responses are

	$ITAE$		E_u	
	MPC	**Cascade**	**MPC**	**Cascade**
$w_0 = 0.5$	816	4896	264	253
$w_0 = -0.5$	803	5053	252	246
$w_0 = 1$	196	3492	377	376
$w_0 = -1$	2415	6902	204	201

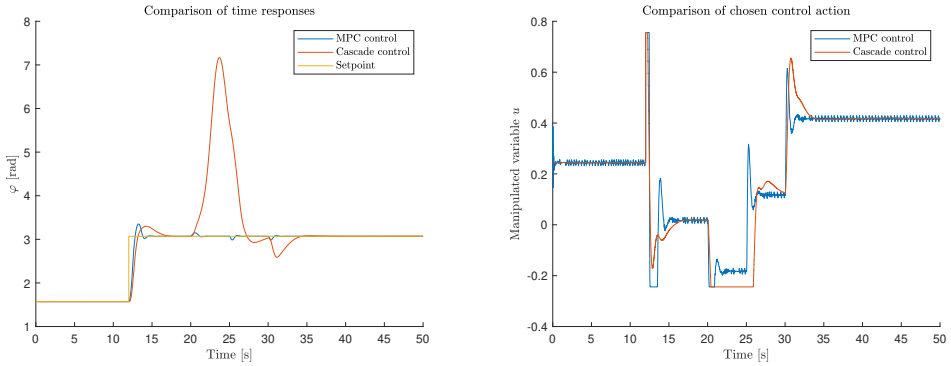
The measurement noise can be seen on the MPC control output more easily than on the cascade control output. This is because the cascade control quantizer is applied on the velocity, rather than on the position like in the case of the MPC controller. Also, the cascade loop also includes a lowpass filter after the quantized measurement. This means that the effect of quantization is stronger for the MPC controller.
The $u(t)$ control of MPC can be filtered more by slowing the speed of estimation. This option should be considered if the noise on $u(t)$ induces an unmodeled high frequency response from the real-world system.

In all of these scenarios, the MPC controller is able to reject disturbances with much higher speed and with less overshoot while still only moving within the constraints put on the MPC output.
The cascade control is significantly slower, taking more than 10 seconds to stabilise the helicopter. The MPC controller is more aggressive, which allows it to reach its target much faster, but this aggressiveness could excite some of the behaviour of the real world system that we did not model, as it would be prohibitively difficult to do so. Thankfully, the MPC controller allows us to easily curb its aggressiveness by assigning a higher weight to changes in control in the cost function.

Comparison with extreme input disturbances present We performed a simulation where the controlled plant is the nonlinear system model. The point of this simulation is to show how the MPC controller is able to function in extreme circumstances better that the classical cascade control.
The applied disturbance was in the form of

$$z(t) = \begin{cases} 0 & t \leq 20 \\ 0.2 & 20 \leq t \leq 25 \\ -0.3 & 25 \leq t \leq 30 \\ -0.3 & 30 \leq t \end{cases} \tag{7.56}$$

For this comparison we used a saturation block on the cascade controller output as that is equivalent to the saturation that would be present in the real-world system.



(a) $y(t)$ (b) $u(t)$

The simulation shows how the cascade control was not able to reject the disturbance fast enough, causing the pendulum to fall over and then swing back upwards. The successful rejection at the end was more a matter of chance on the part of the cascade control rather than of a controlled effort. The cascade control does not deal well with the constraints present causing its behaviour around saturation to be very aggressive. This is mainly due to the low amplification of the outer P control loop. Increasing the amplification causes the loop to be have faster dynamics that are able to reject the input disturbances present in this simulation. This however comes at a cost of a decreased gain margin.
On the other hand the MPC controller was able to reject the disturbance almost without it being visible on the output. It is able to function within the set constraints very naturally, which is one of its main strengths over classical control schemes.

7.6 qpOASES

qpOASES is an open source package written in C++ designed for solving quadratic programming problems. qpOASES was designed specifically for solving QP problems that arrise from MPC controllers. It uses the so called active-set method and offers faster computation speed than the standard *quadprog()* solver. The package is described in [6] (package is availiable at https://github.com/coin-or/qpOASES). qpOASES offers a Matlab interface using MEX functions.

The package includes two main ways to solve QP problems. The standard *qpOASES()* function and *qpOASES_sequence()*. The *qpOASES()* functions in much the same way as *quadprog()*, solving the QP problem from scratch at each sample time.

The *qpOASES_sequence()* implements a so called hot-start variant designed to be utilised mainly for MPC purposes. Since the positive-definite matrix $\mathbf{G}$ does not change over time, the *qpOASES_sequence()* performs one slower computation at the beginning and then uses the previous results to compute the following solutions. This can potentially speed up the computation time for very complex QP problems, but offers only marginal improvements for simple ones.

The article [4] performs numerical comparisons of computation speed between various QP solvers including *quadprog()* and qpOASES for MPC problems similar to those discussed here. We perform comparisons of computation speed between these solvers on our particular system.

We ensured used the qpOASES option "MPC" to tell the qpOASES algorithm to use parameters tuned for general use in MPC applications. These options include the numerical tolerances used to search for the QP optimum and influence the computational time needed. The MPC option uses the fact that MPC QP problems are always strictly convex and disables some of the more costly algorithmic features to ensure the computation is carried out as fast as possible.

7.6.1 Comparison between quadprog and qpOASES

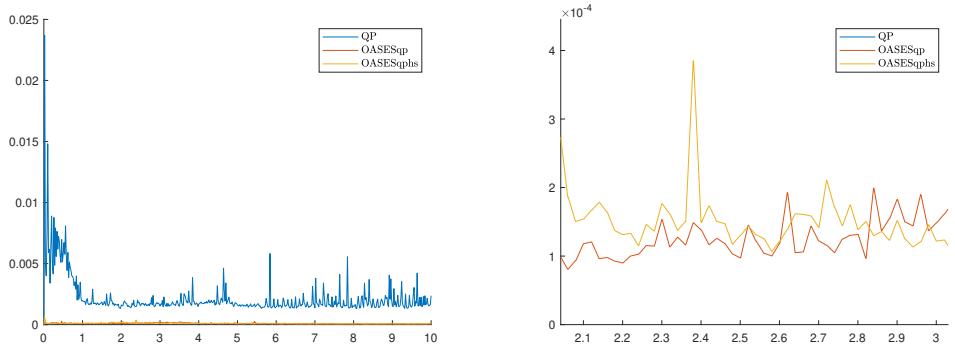
We performed simulations to determine the computational time needed for each solver (quadprog, qpOASES, qpOASES_sequence) to compute the MPC control action at each time step over the simulation duration. This comparison is important because we aim to use the MPC controller in online mode, meaning the controller needs to solve the QP problem at each sample time faster than the next the sample comes. If it cannot compute fast enough the controller becomes essentially unusable.

When tuning the MPC controller we have 4 main parameters that influence the computation time. The prediction horizon n_c, the control horizon n_c, the MPC sample time T_s and the chosen QP solver. We can use the first three to directly improve the MPC performance at the cost of longer computation time. If we were not able to run the MPC controller online using our chosen horizons, we would be forced to choose a slower sample time. We can circumvent this issue by using a better solver.

To perform the comparisons we took 4 measures to gauge the computational intensity. The total time used for computing during the simulation in seconds (Total time). The longest time needed to compute the solution over the samples (Max time). The mean time needed to compute a solution (Mean time). The number of times the solver took longer to compute the solution than is the chosen MPC sample time T_s. And the number of times the solver took longer to compute the solution than a tenth of the sample time T_s.

The simulations for each horizon value were performed 10 times in succession and then the computational times were averaged. This was done to prevent any sudden slowdown caused by the operating system the simulations took place on. The simulations were performed on a Windows 10 machine with Intel(R) Core(TM) i5-8300H CPU @ 2.30GHz.

Comparison for $n_p = 50, n_c = 10$ Simulations were performed to estimate the computational time for $n_p = 50, n_c = 10$. These horizon values are conservative estimates. But were used for the comparisons between MPC and PID, meaning they are very much usable.



(a) Simulation time

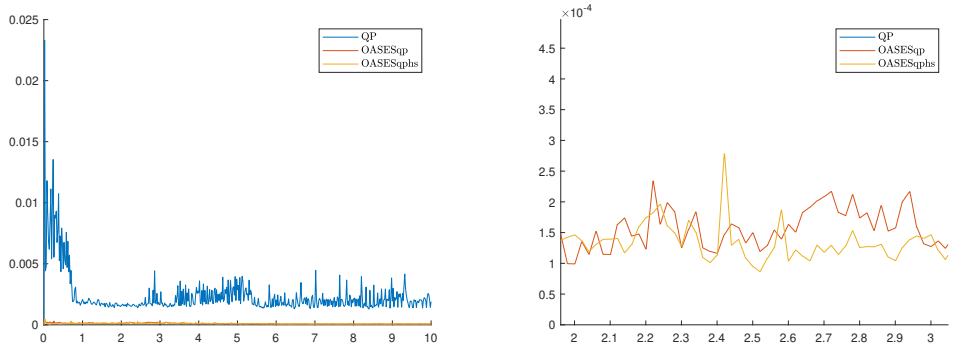
(b) Close up

Figure 43: Computation time per sample over simulation time for $n_p = 50, n_c = 10$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	2.1737	0.0237	0.0022	2	253
qpOASES	0.0955	0.0005	0.0001	0	0
qpOASES hotstart	0.0931	0.0006	0.0001	0	0

Table 2: $n_p = 50$, $n_c = 10$

Comparison for $n_p = 100, n_c = 20$ Simulations were performed to estimate the computational time for $n_p = 100, n_c = 20$.
All three algorithms are almost identical in its computational time when compared to the previous scenario. This is likely because the bulk of their computational time is spent on overhead for horizon values so small.



(a) Simulation time

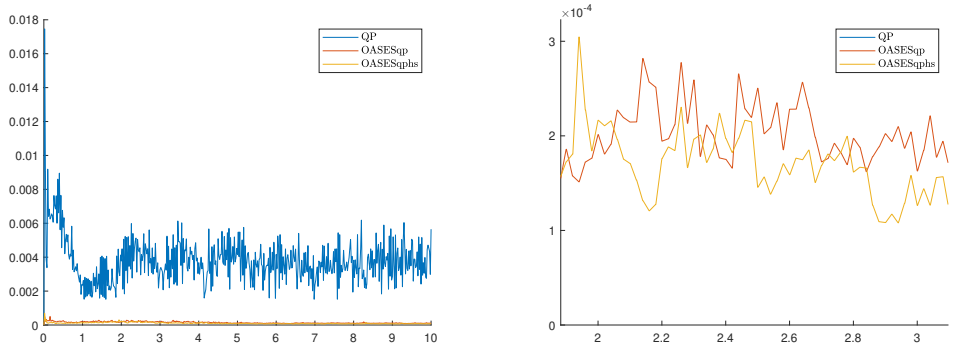
(b) Close up

Figure 44: Computation time per sample over simulation time for $n_p = 100, n_c = 20$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	2.3881	0.0233	0.0024	2	0
qpOASES	0.1048	0.0005	0.0001	0	0
qpOASES hotstart	0.0929	0.0005	0.0001	0	4

Table 3: $n_p = 100$, $n_c = 20$

Comparison for $n_p = 200, n_c = 40$ Simulations were performed to estimate the computational time for $n_p = 200, n_c = 40$.
Here we can see that the quadprog algorithm takes longer than tenth of the sample time for almost all samples. This is not an issue in and of itself, but is a clear indication that if we increase the horizons further, quadprog will not be able to keep up. The qpOASES algoritms keep way below the $\frac{T_s}{10}$ threshold.



(a) Simulation time

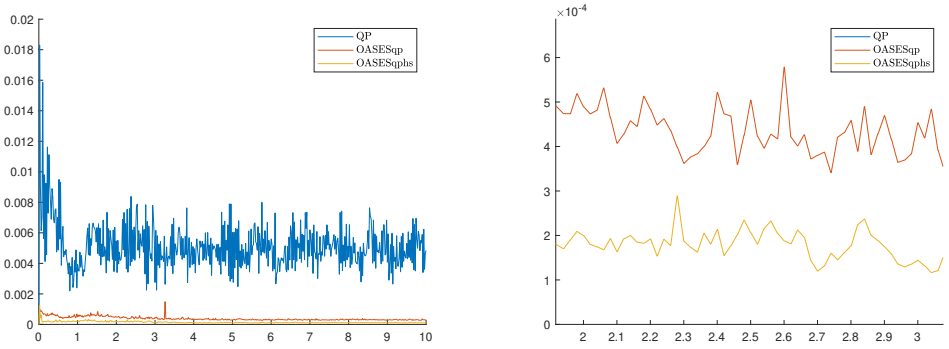
(b) Close up

Figure 45: Computation time per sample over simulation time for $n_p = 200, n_c = 40$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	3.7868	0.0175	0.0038	0	946
qpOASES	0.1491	0.0005	0.0001	0	0
qpOASES hotstart	0.1069	0.0007	0.0001	0	0

Table 4: $n_p = 200$, $n_c = 40$

Comparison for $n_p = 400, n_c = 80$ Simulations were performed to estimate the computational time for $n_p = 400, n_c = 80$.
This simulations shows the trend of quadprog needing more and more time to complete, with 0.0183s being the max time needed to perform one computation. This is 10 times longer than qpOASES. We can also see that while almost all time samples for quadprog took longer than $\frac{T_s}{10}$, they never took longer than T_S. This means that while computationally intensive, quadprog computation time is relatively stable.



(a) Simulation time

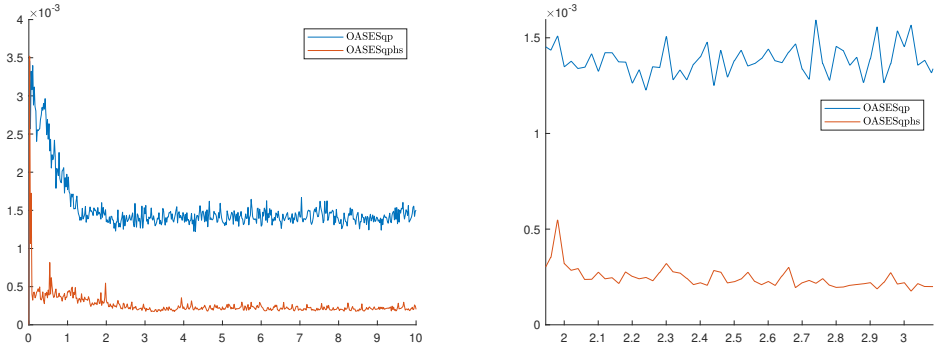
(b) Close up

Figure 46: Computation time per sample over simulation time for $n_p = 400, n_c = 80$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	5.1249	0.0183	0.0051	0	999
qpOASES	0.3835	0.0015	0.0004	0	0
qpOASES hotstart	0.1428	0.0013	0.0001	0	0

Table 5: $n_p = 400, n_c = 80$

Comparison for $n_p = 800, n_c = 160$ Simulations were performed to estimate the computational time for $n_p = 800, n_c = 160$.
For this simulation we opted out of including quadprog, as it was obvious that it would not be able to keep up with the online computational requirement.
Here we can see that while before, qpOASES and its hot-start variant were comparable with one another. The standard variant is getting slower faster. This is to be expected as the optimalization offered by the hot-start variant is more effective for larger QP problems. Here, the total time needed to compute the hot-start variant was 5 times less than the standard qpOASES solver.



(a) Simulation time

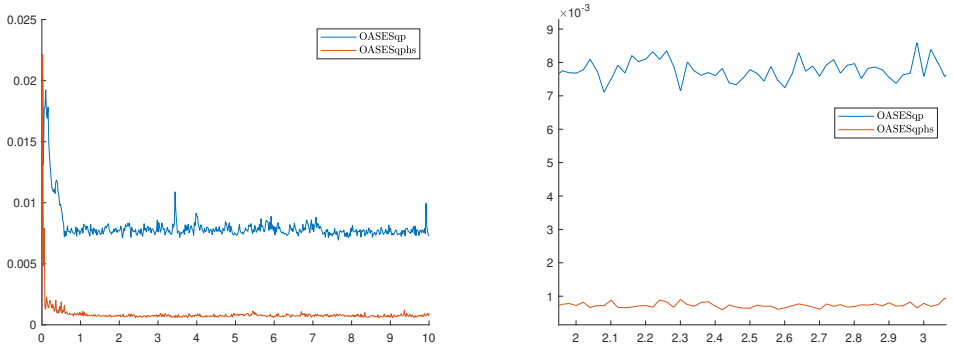
(b) Close up

Figure 47: Computation time per sample over simulation time for $n_p = 800, n_c = 160$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	-	-	-	-	-
qpOASES	1.5211	0.0035	0.0015	0	80
qpOASES hotstart	0.2586	0.0035	0.0003	0	2

Table 6: $n_p = 800, n_c = 160$

Comparison for $n_p = 1600, n_c = 320$ Simulations were performed to estimate the computational time for $n_p = 1600, n_c = 320$.
The trend here continues, as the hot-start variant is 10 times faster than the standard qpOASES solver, which is starting to struggle with the computational demands.



(a) Simulation time

(b) Close up

Figure 48: Computation time per sample over simulation time for $n_p = 1600, n_c = 320$

	Total time	**Max time**	**Mean time**	**Over T_s**	**Over $\frac{T_s}{10}$**
quadprog	-	-	-	-	-
qpOASES	8.0393	0.0203	0.008	2	999
qpOASES hotstart	0.8558	0.0221	0.0009	2	10

Table 7: $n_p = 1600, n_c = 320$

7.6.2 Comparison conclusion

From these simulations we found out that the Matlab QP solver quadprog is significantly slower than the open-source alternative qpOASES. When available, qpOASES should be used instead of quadprog, since it offers the same structure, while being more efficient.
The hot-start variant of qpOASES was also measured. Its computational time was generally shorter than qpOASES, especially for large scale QP problems.
Since we are getting satisfactory results from the MPC controller when using the least performance heavy horizons, we need not worry about the solver used too much. But moving further, we will only use the standard version of qpOASES as it allows us plenty of breathing room while minimizing the additional overhead needed to use the hot-start variant.

8 Conclusion

A mathematical model of the Humusoft CE150 helicopter was developed using first principles. An experiment was performed on the model helicopter to gather data for measurement. This data was then used to numerically compute its first and second derivatives which were then filtered using a FIR zero-phase discrete filter. The model was then identified using the least squares regression method.

An model-predictive controller was designed and implemented using Matlab and Simulink to control this mathematical model. This thesis shows that MPC is a viable method of controlling the Humusoft CE150 model helicopter. The MPC controller outperforms classical cascade control when following a piece-wise reference signal and when rejecting input disturbances. It is able to apply the appropriate control law while satisfying the constraints we set on the system thanks to the limited saturation ranges of the physical actuator.

A open source alternative qpOASES to the Matlab quadprog was found to be significantly faster when computing the MPC control. This finding shows us that the MPC controller will be able to be used to control the helicopter model in online mode, as we are able to reach satisfactory control even when using computationally unintensive MPC parameters.
Moving further, the MPC could be upgraded to include constraints on the pitch angle of the helicopter that correspond to the physical barriers on the real-world model.
The MPC controller was not tested on the real-world helicopter. All the results of this thesis were done using only computer simulation. The next developmental step is to implement the MPC control algorithm in REXYGEN and then attempt to apply it on the helicopter model using a microcomputer.

References

[1] Paul Serban Agachi, Mircea Vasile Cristea, Alexandra Ana Csavdari, and Botond Szilagyi. *2. Model predictive control*. De Gruyter, 2016.

[2] Edin Dragolj, Jasmin Velagic, and Nedim Osmic. Modelling of nonlinear helicopter model and loopshaping based controller synthesis. pages 3603–3608, 11 2013.

[3] Kristofer Jennings. Statistics 512: Applied linear models. *Internet.[Google Scholar]*, 2012.

[4] Ondřej Mikuláš. Quadratic programming algorithms for fast model-based predictive control. *Copyright©[cit. 27.05. 2020]. Dostupné z: https://support. dce. felk. cvut. cz/mediawiki/images/9/94/Bp _*, 2013.

[5] J.A. Rossiter. *Model-Based Predictive Control: A Practical Approach*. CRC Press, 2003.

[6] Hans Joachim Ferreau, Christian Kirches, Andreas Potschka, Hans Georg Bock, and Moritz Diehl. qpoases: A parametric active-set algorithm for quadratic programming. *Mathematical Programming Computation*, 6(4):327–363, 2014.