NTNU – Trondheim
Norwegian University of
Science and Technology

Heave disturbance attenuation in managed pressure drilling from a floating platform using model predictive control

Edvin Hatlevik

Master of Science in Engineering Cybernetics (2 year))
Submission date: June 2014
Supervisor: Tor Arne Johansen, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

NTNU – Trondheim
Norwegian University of
Science and Technology

Heave disturbance attenuation in managed pressure drilling from a floating platform using model predictive control

Edvin Hatlevik

June 2, 2014

Prosjekt oppgave

Institutt for Teknisk Kybernetikk

Norges teknisk-naturvitenskapelige universitet

Supervisor: Professor Tor Arne Johansen

Problem Description

Managed pressure drilling has received increased attention due to its ability to perform drilling with small pressure margins. A significant challenge in offshore operations from a floating platform is the effect of the platform´s motion. During pipe connection the conventional heave compensation system is not operational as the drill string must be clamped to the drill floor. Consequently, the drill bit acts as a moving piston near the bottom of the well such that motion of the platform leads to large pressure variations at the bottom hole, potentially causing damage to the well or risk for well control issues. The task is to study how an MPC can exploit motion predictions to compensate for the pressure variations. And how MPC for managed pressure drilling can be implemented in an industrial PLC. These steps are gone through to study and test how an MPC can exploit motion prediction to compensate for the pressure variations.

1. Find a way to linearize the model, and discretize the model using existing schemes for discretization.

2. Formulate a linear MPC for control of bottom hole pressure, using the choke as manipulated variable, with feedforward from measured and future predicted heave motion.

3. Design a state estimator based on Kalman-filtering.

4. Use a dynamic model to simulate the motion of a semi-submersible drilling rig to test the MPC in Matlab.

5. Design an MPC implementation suitable for implementation on an industrial PLC, and test using simulations.

Summary

Since a large part of the Norwegian oil shelf has been active for over a generation, many fields begin to be depleted and the drilling operations requires tight down hole pressure margins. And by improving the pressure control for the drilling operations former undrillable wells becomes drillable. Which will make the the oilfields more profitable, and extend their life expectancy. It will also make drilling operations safer by preventing kicks and preventing environmental damages caused by mud leaking into the pore space.

One of the most critical phases when drilling from a floating drilling rig in terms of down hole, is pipe connection. During this procedure the conventional heave compensation is not operational as the drill string is climbed to the drill floor. Consequently the drill bit functions as a piston creating large pressure variations in the drill bit pressure. To control this pressure and create disturbance attention a linear model predictive controller with feedback linearisztion is created using feedback linearization. This controller shows promising results when feedforward with future predictions is applied. Without future predictions the results are the same as for an MPC without feedforward. Because only the topside pressure is known the rest of the states are estimated using a Kalman filter, which shows good results on the state estimations. To make the system more applicable in real life applications an efficient linear model predictive controller implementation was created for a PLC with great results, both in terms of calculation time and memory usage.

Preface

This master thesis was written in my 4th and final semester at NTNU for the MSc. degree in Engineering Cybernetics. It was carried out in the period from January to June, 2014. I was accepted into this master's program based on my BSc. degree in Automation from Ålesund University College, from 2012. The assignment has been carried out under the supervision of Professor Tor Arne Johansen.

I would like to tank my advisor Tor Arne Johansen for his work and contribution to this thesis, which is greatly valued. I would also like to thank Kwame Minde Kufoalor for his support regarding, MPC implementation on PLCs.

Trondheim, 2014-06-02

Edvin Hatlevik

Abbreviations

BHA	Bottom Hole Assembly
Cvr	Controlled Variable
DHP	Down hole pressure
Dvr	Disturbance variable
IRIS	International Research Institute of Stavanger
KKT	Karush-Kuhn-Tucker
MPC	Model Predictive Control
MPD	Managed Pressure Drilling
Mvr	Manipulated Variable
NMPC	Non-linear Model Predictive Control
PID	Proportional Integral Derivative
QP	Quadratic Problem
SISO	Single Input Single Output
SQP	Sequential quadratic programming

Contents

Chapter 1

Introduction

1.1 Norwegian oil

Until the late 1950s very few people believed that there where petroleum in the north sea. But in 1959 a Dutch gas field was found in Groningen. This raised some attention to the north sea, and some experts began speculating if there could be oil in the Norwegian sea. Norwegian geologist was more negative, and did not believe there was any oil or gas in the Norwegian sea. This did not stop the enthusiasm, and in October 1962 the Norwegian government received a leather from Phillips petroleum, where they wanted permission to start oil exploration on the Norwegian shelf. The company wanted a license on the Norwegian shelf, and offered 160 000 USD a month for such a deal. The Norwegian government declined the deal, because they felt that Phillips wanted exclusive rights. In 1963 the Norwegian government wrote a law making the king(government) landowner of the hole Norwegian shelf. Two years later, in 1965 a deal was made with Great Britain and Denmark regarding the sharing of the north sea, where the median line principle was used. One year later in 1966 the first Norwegian exploration well was drilled, it was empty. The first oil was found in 1969 by Phillips petroleum approximately 320 km south west of Stavanger. It was called Ekofisk and is still one of the most important oil findings in the north sea. This has obviously created huge incomes to the Norwegian government, and 22.June.1990 the Norwegian government chose to save the income in a fund (Statens petroleumsfond) to stabilize the economy. This has of course done more then stabilize the economy because in the end of 2013 the fund had a total value of 850 billion USD. In the early north

see drilling operations it become clear that drilling oil in the north sea had technological difficulties both in the terms of weather and depth. Which lead to an increased activity in Norwegian oil and gas related research. For more information about the Norwegian oil history, visit the Norwegian governments web pages.

www.regjeringen.no/en/dep/oed/Subject/oil-and-gas

In the beginning of the Norwegian oil adventure the wells where drilled straight down. But as time passed there began to be an increasing demand for more advanced drilling operations. The need for advanced drilling operations is due to the easily accessible petroleum becomes dried up, and to get the less available petroleum pockets more advanced techniques are used. This also makes it possible to drill into several petroleum pockets in different locations from the same oil installation. Which of course is profitable because fewer oil installations needs to be built.

1.2 Model Predictive Control in Managed Pressure Drilling

One of the aspect of these new more advanced drilling techniques is that there is higher demand for pressure control while drilling. There is many reasons why the drilling pressure is important. Logically one would maintain the down hole pressure between a minimum and a maximum value. Where the limits are imposed by the formation of the well and other geological factors. To make it more tangible, here are some examples of what might occur if these limits are exceeded.

To low pressure	To high pressure
Gas might leak in to the mud creating extreme pressures called kicks	Mud leaks into the formation
Drilling into neighbouring wells.	Mud forms a wall over the pores in the well, slowing the production.
Getting influx of oil or water into the mud	Overburdening the well by applying more pressure than the combined weight of the overlaying formation
Well collapses around the drill string	

This is of course a bit shortened and it was just meant as a motivation to why pressure control in drilling operations is important.

Now back to the case at hand, keep the pressure between the limits. To do this a control strategy was developed called Managed pressure drilling(MPD) or more precise a variant called constant bottom hole pressure. Where the principle is to control the bottom hole pressure when a back pressure is applied with the choke valve. To control the bottom hole pressure a controller is applied to the choke valve. An example of such a controller might be a PID controller. Much research has been devoted to MPD control the last decade, not only internationally but also in Norway. An early example of such research is (Johan Eck-Olsen, 2005) where they used managed pressure to cement a seven inch liner on the Gullfaks field. Or more resent development in (Godhavn, 2010) where some requirements for high-performance control where presented.

Another approach that has been discussed by control engineers in the oil industry for some time, is to use model predictive control to control the bottom hole pressure. Which is a specific type of advanced control theory. The reason why this controller type is being looked at besides it's excellent control results, is it's native ability to handle constraints. Where constraints in the MPD control case, refers to the desired limits on the down hole pressure, and the open and closed limitations on the choke valve.

The master thesis (Øyvind Breyholtz, 2008) is exploring the use of non-linear model predictive control with a combination of measurements and estimation for acquiring readings of the states. Where the bottom hole pressure is the controlled variable and the choke vale the manipulated variable, can be found at (Øyvind Breyholtz, 2008).

A master thesis, studying linear model predictive control with assumed measured down hole pressures, where three different down hole pressures are controlled variables and the manipulated variables are mud pump, choke and back pressure pump can be found at (Møgster, 2013).

1.3 MPC For Heave Disturbance Attenuation in MPD systems

When designing managed pressure drilling control systems, one should not just design a system for an ordinary drilling application, but take into account for optional procedures and disturbances that affect the system performance. A specific such pressure is when extending the drill string in order to drill deeper. During this pipe connection the drill string is clamped to the drill floor, leaving the conventional heave compensation system inoperable. In this case the rig moves vertically with the waves, referred to as heave motion. Because the drill string is clamped to the drill floor, this motion will translate into a drill bit moving like a piston on the mud in the well. Consequently creating severe pressure fluctuations at the bottom of the well. These fluctuations have been observed to have a magnitude higher than the standard limits for pressure regulation accuracy in MPD, which is about ±2.5[bar] (Amirhossein Nikoofard and Pavlov, 2014). When the drill bit moves down into the well the pressure increases (Surging), and upward movement decreases the pressure (swabbing). Strong surging and swabbing pressures can cause damage to the well, neighbouring wells, personnel, environment or drilling equipment as mentioned in the previous section.

In the article (Ingar Skyberg Landet, 2013) a semi linear dynamical model was created to capture the main dynamics of a MPD system in the case of heave disturbance in a well from the Ulrig test drilling facility, with a length of approximately 2000 m with water based mud. This model contained nine ordinary differential equations, and a choke equation. They also presented two controllers for disturbance attenuation, which successfully damped the disturbance.

Amirhossein Nikoofarad later used this model in (Amirhossein Nikoofard and Pavlov, 2014) to create a MPC controller with feedforward for disturbance attenuation.

In this master thesis, applied backpressure managed pressure drilling will be the choice of set up in combination with a linear feedforward model predictive controller, using the model from (Ingar Skyberg Landet, 2013). Designed for heave disturbance attenuation. Down hole measurements will be estimated using a Kalman filter. The manipulated variable will be the choke valve.

The controlled variable will be the drill bit pressure. Where the constraints imposed on the manipulated variable is denoted by the limitations on the choke valve, in terms of open closed valve. The constraints imposed on the controlled variable is denoted from the pressure limitations of the well. This controller will then be implemented in an industrial PLC, using an efficient C code solver generated by the ACADO toolbox. To the authors knowledge these controllers are usually implemented using industrial computers. This is supposed to be a novel approach to make these types of controllers easier to implement on existing installations, preferably using equipment already installed.

1.3.1 Research Focus

- At first the focus will be on creating a working MPC controller, and an estimator suitable for PLC implementation. The next step will be to optimize this controller to be as lightweight as possible, and run as swiftly as possible, because the PLC has a lack of both processing power and memory.
- Another aspect that need some attention is the implementation of constraints on the manipulated variable, because they are non-linear.
- Testing different ways of implementing feedforward, and the effect on the disturbance attenuation.
- Find a way to implement the controller on the PLC, and evaluate the performance.

1.3.2 Thesis Outline

The outline of this thesis is as follows. Chapter 2 is devoted to drilling in general and managed pressure drilling. Chapter 3 introduces heave disturbance, modelling of this disturbance and state estimation. Chapter 4 introduces model predictive control and the different properties of such controllers. Chapter 5 presents the key aspects of designing the model predictive controller and specific data used in Chapter 6 and 7. Chapter 6 goes more into specific details on implementation of the controller. Chapter 7 builds up scenarios for simulations and simulates these scenarios.

Chapter 2

Drilling Systems

In figure 2.1 one can see an illustration of a floating drillrigg with a setup for managed pressure drilling, where the important components are outlined.

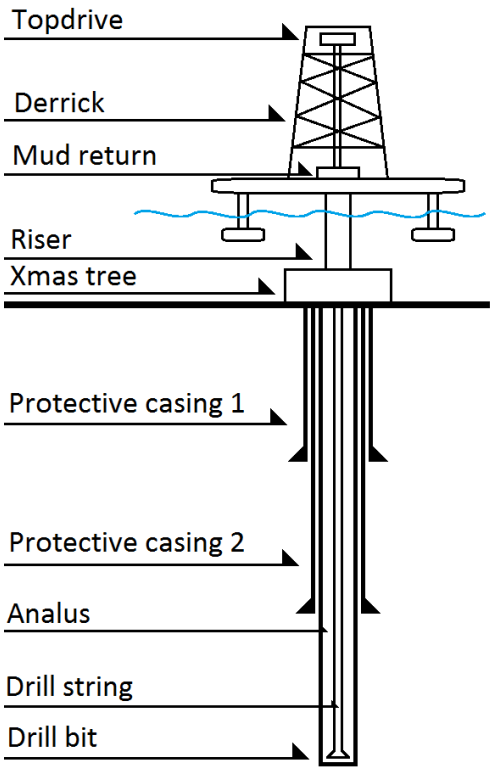


Figure 2.1: Drilling operation

The derrick or the drill tower is where the drill string is assembled. And on the top of the derrick the drill string is connected to the top drive. The topdrive holds the engine that turns the drill string and it is where the mud is pumped in to the drill string. The drill string and the topdrive can be moved up and down on the derrick as one drills deeper. Each drill string is approximately 9 meters long and are connected into a 27 meters long drill stand. An new drill stand is connected to the top of the last each time the last is drilled into the ground. It takes approximately two hours to drill a drill stand int the ground which means a new needs to be added every two hours at usual drill speed. The drill strings motion in the derrick is also used to compensate for the motions of the drilling rig due to waves. When new pipes are connected to the drill string, the string needs to be clamped to the drill floor. This causes the drill bit to act as a piston near the bottom of the well.

Mud is pumped from the mud pit through the top drive, down the drill string, and out the drill bit. The mud then brings along the drill cuttings from the drilling operation up the annulus , as shown in figure 2.2. From the top of the annulus the return mud flow q_c is controlled by the choke valve. Then purified in the shale shakers before it is transported back to the mud pit. The mud plays many important roles in the drilling process, and in the list below some of its major responsibilities are mentioned.

1. When drill cuttings needs to be transported from the drill hole up the annulus and separated from the mud in the shale shakers. This require the mud to have rather high viscosity in order to be able to bring the cuttings along. This is described in more detail in (Øyvind Nistad Stamnes, 2011).

2. During drilling the drill bit may overheat and the mud acts as a coolant for the drill bit.

3. The flow rate back is used under managed pressure drilling to control the drill bit pressure p_{bit}.

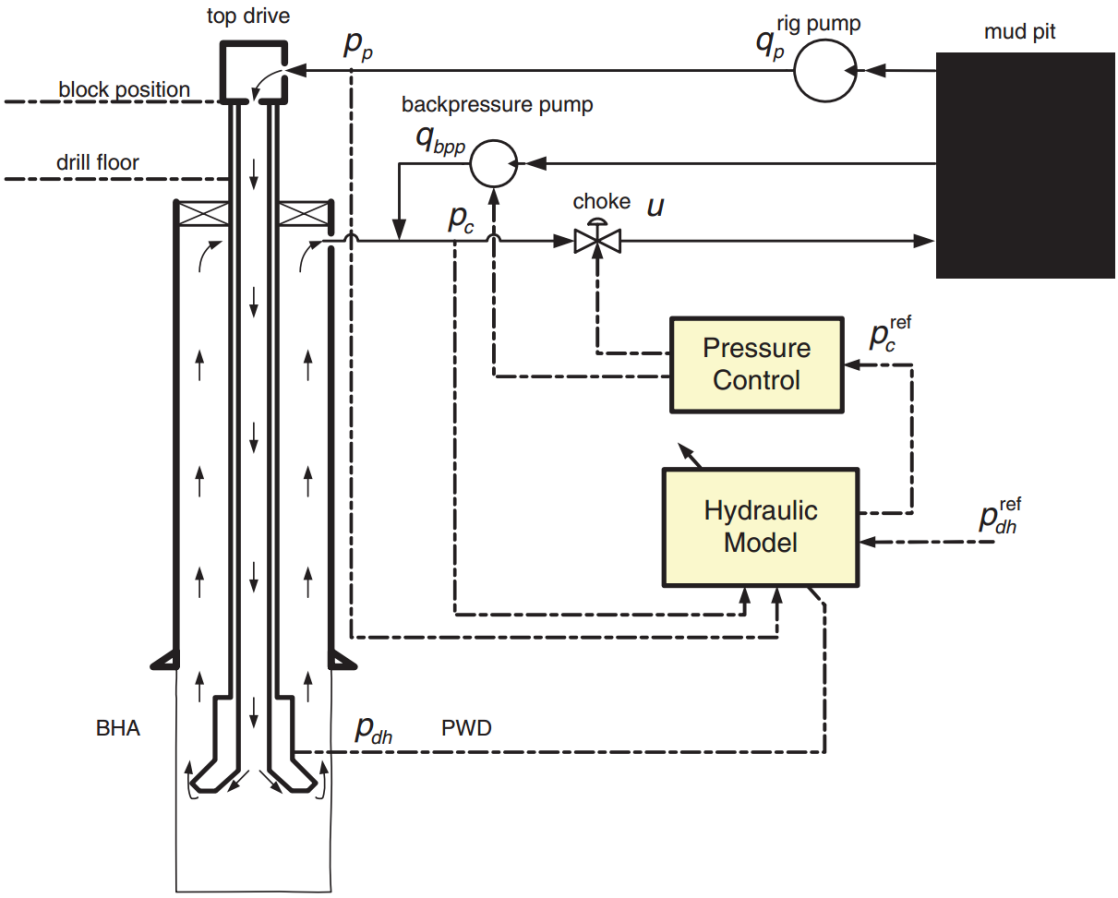


Figure 2.2: Schematic of an automated ABP-MPD system. Copied from (Kaasa, 2012)

2.1 Pore and fracture pressure

All formation penetrated by the drill bit is to some extent porous and may contain oil, gas or salt water. These fluids contained in the pore space builds up the pore pressure, p_{pore}. It is important to keep the drill pressure p_{bit} higher than the pore pressure. If this criteria is not upheld one of the following situations may occur.

1. If the pressure and the viscosity is to low, gas might leak into the mud creating so called kicks. It's called kicks because the gas expands as it rises to the surface and creates dangerous increase in pressure and can cause a blow-out.

2. Drilling into neighbouring producing wells.

3. Getting influx to the mud in form of oil or water. This is only wanted during production and may cost both production value and affect the muds viscosity.

If on the other hand the drill pressure gets higher than a certain pressure p_{frac} which will cause the mud to leak in to the formation, one may risk that.

1. Mud leaks into the pore space, causing mud loss. This is both costly and in violation of environmental laws.

2. Mud might form a wall over the pores in the drill hole. This problem can be solved by re-drilling the area, or it will slow down production.

In short the drill pressure must be held within the pressure parameters.

$$p_{pore} < p_{bit} < p_{frac}$$

Beyond these boundaries are some worst case boundaries.

$$p_{collapse} < p_{pore} < p_{bit} < p_{frac} < p_{overburden}$$

The $p_{collapse}$ is the pressure limit where pressure becomes so low that the well will collapse around the drill string. Consequently the drill string may be stuck in the drill hole. $p_{overburden}$ is the combined weight of formation materials and fluids in the geological formations above any particular depth of interest in the earth (Skalle, 2011). Lastly it is worth mentioning that the inequality above, is not completely written in stone, the $p_{collapse}$ might be larger then the pore pressure in some rare occasions.

The pore and fracture pressure is calculated by geologist prior to the drilling, and can be verified during drilling operations. In figure 2.3 one can see a representation of the possible pressure limits imposed by the pore and fracture pressure, and how the pressure limits is changing with the depth of the well. One of the control objectives are to contain the down hole pressure between the pressure limits calculated by the geologists.

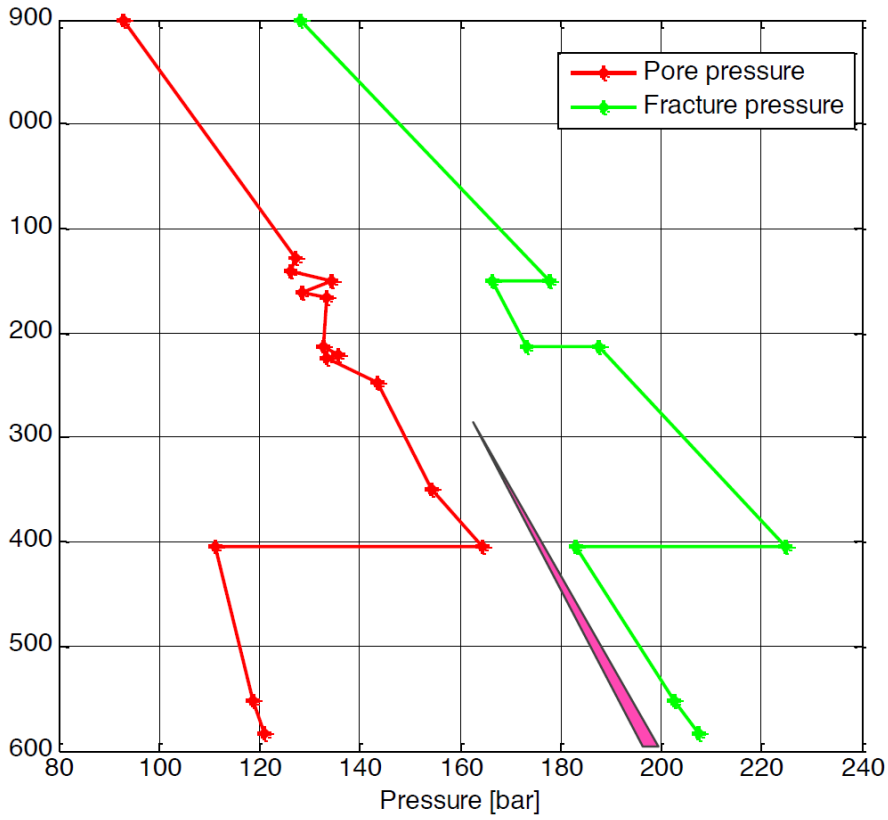


Figure 2.3: Pore and fracture pressures at given depths. Copied from (Kaasa, 2012)

2.2 Managed pressure drilling

As one can imagine from the last section, there has become a high demand for accurate control of the pressure in annulus during drilling operations. This has led to the rise of Managed pressure drilling which is best described in the capable hands of (Øyvind Nistad Stamnes, 2011).

> *"Managed Pressure Drilling is an adaptive drilling process used to precisely control the annular pressure profile throughout the wellbore. The objectives are to ascertain the down hole pressure environment limits and to manage the annular hydraulic pressure profile accordingly. The intention of MPD is to avoid continuous influx of formation fluids to the surface. Any influx incidental to the operation will be safely contained using an appropriate process."*

—Øyvind Nistad Stamnes (2011)

When we talk about managed pressure drilling(MPD), we usually mean an adoption where the bottom hole pressure p_{bit} is desired to be constant. This is a principal called the constant bottom hole pressure. Were one wants to keep the bottom hole pressure between the pressure limits described in the section above.

A Simple MPD system is illustrated in figure(Kasaa).Were the control goal is to keep the drill bit pressure p_{bit} stable. This is done by controlling the feedback flow through the choke valve. In this thesis the main focus will be on the part of the drilling operation where the drill string is bolted to drill floor for pipe jointing. And due too this the flow through the top drive $q_p = 0$. As a result of this the drill bit pressure has to be maintained by the back pressure pump. Because most measurements are taken top side one usually do not have the luxury of measuring any other states than the top side pressure p_c. The rest of the states including p_{bit} which is vital for pressure control has to be estimated.

2.3 Hydraulic model

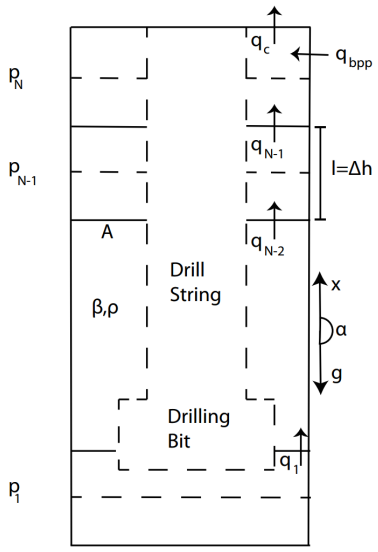


Figure 2.4: Control volumes of the annulus hydraulic model. Copied from (Ingar Skyberg Landet, 2013)

In (Ingar Skyberg Landet, 2013) they created a hydraulic model with five control volumes to capture the main dynamics of the ullrig test drilling facility. The model is made for controlling the bottom hole pressure $p_{bit} = p_1$ when a new drill string is being mounted. This results in a model that dose not have the mud pump flow in the dynamics because the mud flow through the top drive $q_p = 0$. Consequently the only volumetric flow to create a down hole pressure becomes the feedback flow q_{bpp} generated by the back pressure pump.

This resulting hydraulic model is split into five control volumes where each volume has a differential equation denoting the pressure in this volume and a differential denoting the volumetric flow rate from this volume to the volume above. Each of these control volumes has a volumetric flow rate into control volume and out of the control volume. Where the flow from the up most control volume is determined by the flow through the choke valve q_c, and the flow into the lower

control volume is created by the drill bit motion $v_d A_d$.

$$
\begin{aligned}
\dot{p}_1 &= \frac{\beta_1}{A_1 l_1}\left(-q_1 - v_d A_d\right) \\
\dot{p}_2 &= \frac{\beta_2}{A_2 l_2}\left(q_1 - q_2\right) \\
\dot{p}_3 &= \frac{\beta_3}{A_3 l_3}\left(q_2 - q_3\right) \\
\dot{p}_4 &= \frac{\beta_4}{A_4 l_4}\left(q_3 - q_4\right) \\
\dot{p}_5 &= \frac{\beta_5}{A_5 l_5}\left(q_4 - q_c + q_{qpp}\right) \\
\dot{q}_i &= \frac{A_i}{l_i \rho_i}\left(p_i - p_{i+1}\right) - \frac{F_i(q_i) A_i}{l_i \rho_i} - A_i g \frac{\Delta h_i}{l_i} \\
q_c &= K_c \sqrt{p_5 - p_0} G(u)
\end{aligned}
\tag{2.1}
$$

Where $i = 1, \ldots, 4$ and the lower-case numbers refer to the control volume. And the parameters are defined as.

- β_i: The bulk modulus of the mud in control volume i.
- A_i: Is the cross section area in control volume i.
- l_i: Length of each control volume i.
- Δh_i Height of each control volume i.
- ρ_i: Mud density in control volume i.
- $F_i(q_i)$: Friction force in the control volume i.
- g: Acceleration of gravity.
- p_0: Atmospheric pressure.
- K_c: Choke constant.
- v_d: Heave velocity due to ocean waves.

Where control volume $i = 1$ refer to the lower control volume, which means that $p_1 = p_{bit} = p_{dh}$. Because there are five control volumes $p_5 = p_c$ becomes the downstream choke pressure, and q_c

the choke volumetric flow rate. In principle you could control the down hole pressure both with the back-pressure pump and the choke valve. Pump control would require the pump to change speed so fast that it would change the down pressure faster then the waves. This is generally not possible so the choke valve is mainly used for pressure control.

2.4 Instrumentation

Instrumentation is the link between the control system and the process. It's important to understand what we are measuring and what we are controlling in in order to control the process correctly.

2.4.1 Pressure Measurement

Pressure is usually measured in industrial process by a type of instrument often referred to as a pressure transmitter or (PT). These transmitters are mainly categorized into tree different types of transmitters

- **Differential pressure transmitter**

  One of the way to measure pressure is to measure the differential pressure between two spots. This is usually used to measure the level in pressurised tanks as shown in figure DiffPreasure, flow in pipes or filter clogging.

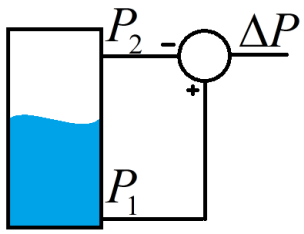


Figure 2.5: Differential pressure transmitter

- **Absolute pressure transmitter**

  This sensor measure pressure relative to perfect vacuum.

$$p_a = p_g + p_{atm}$$

And is usually used if the transmitter doesn't have access to a atmospheric pressure. If this type of transmitter is used it's common practise to use a second absolute pressure transmitter to measure the atmospheric pressure. This way one can abstract the gauge pressure by subtracting the absolute pressure

$$p_g = p_a - p_{atm} = p_g + p_{atm} - p_{atm}$$

- **Gauge pressure transmitter**
  This is probably what one might call a common pressure transmitter. It measures pressure relative to atmospheric pressure. It can be described as a differential pressure transmitter if p_1 is the measurement and p_2 is the atmospheric pressure

And for each of these transmitters a range of different pressure sensing technologies can be used (Capacitive,Piezoelectric,Electromagnetic,etc).

These measurement is then sent to the control system in our case a (PLC). To do this there exists a range of different standards, and one of the most common is 24v, $4-20$mA. Where 4mA is 0% and 20mA is 100% of the measurement scale.

In the MPD system setup used in this thesis, the only available relevant measurement is the top side pressure p_c. This is not enough to control this system, mainly because the controlled variable is the down hole pleasure. But there is also a need to acquire the rest of the states used in the MPC controller. To get the rest of the states the measured pressure is used in a Kalman filter explained in section 3.2 combined with the system model from section 2.3 to estimate the rest of the states.

2.4.2 Choke Valve

To have a control system one must have a manipulated variable. In a MPD systems it's common to control the return flow in order to control the bottom hole pressure. The return volumetric

flow rate q_c is controlled by the choke valve.

$$q_c = K_c \sqrt{p_5 - p_0} G(u) \tag{2.2}$$

Where the choke characteristic $G(u)$ can be defined as a polynomial function

$$G(u) = a_g u^2 + b_g u + c_g$$

which is not necessarily a quadratic function. But in many cases this is a sufficient notation. Another factor that should be taken into consideration that isn't included in the equations is that valves have limits to how fast they can move. The M-I SWACO ECHOKE [swaco] which is advertising for being a fast choke, uses 8 seconds from full open to full close. This means if the controller sets an output to the choke valve it's not realistic to assume that this instantaneously will be the actual choke output.

2.5 State space model

In order to implement the system in a MPC controller the system needs to be linearised and written as a linear sate space model.

Based on experimental research from the Ullrig test data, the friction force in annulus can be considered a linear function.(Ingar Skyberg Landet, 2013)

$$F_i(q_i) = \frac{k_{fric,i} q_i}{A_i}$$

where

$$k_{fric,i} = \frac{64 l_i \mu_i (\alpha_i + \beta_i)}{r_{h,i}^2}$$

where the new parameters are defined as.

- μ_i is the viscosity in the specific control volume.
- α_i and β_i are constants related to the ratio between the diameters of the annulus for in-

struction in calculation see (Ingar Skyberg Landet, 2013).

- r_h: Is the hydraulic radius.

This leads to the linearised flow model

$$\dot{q}_i = \frac{A_i}{l_i \rho_i}\left(p_i - p_{i+1}\right) - \frac{K_{fric}}{l_j \rho_j} q_i - A_i g \frac{\Delta h_i}{l_i}$$

Another non-linearity is the choke characteristic discussed in section 2.4. This one can't be linearised directly but by using feedback linearisation this non-linearity can also be removed in order to create a linear system.

$$\begin{aligned} u_a &= q_{bpp} - q_c \\ &= q_{bpp} - K_c \sqrt{p_c - p_0} G(u) \\ G(u) &= \frac{q_{bpp} - u_a}{K_c \sqrt{p_5 - p_0}} \end{aligned}$$

the equation for the top pressure becomes.

$$\dot{p}_5 = \frac{\beta_5}{A_5 l_5}\left(q_4 + u_a\right)$$

Where the choke characteristics can be approximated as a second degree polynomial.

$$G(u) = g = a_{choke} u^2 + b_{choke} u + c_{choke}$$

Because the input $1 \geq u \geq 0$, the negative root of the answer becomes irrelevant, and the control input becomes.

$$u = \frac{-b_{choke} + \sqrt{b_{choke}^2 - 4a_{choke}(c_{choke} - g)}}{2a_{choke}}$$

How general is nonlinearity cancellation? It is not possible to cancel nonlinearities in all nonlinear systems. In order to cancel this kind of nonlinearities the system must have some structural properties. In order to make this properties more tangible (Hhalil, 1996) have created definition 1 to show that a system is feedback linerizable. This theorem clearly states that the system in

(2.1) must be possible to rewrite into

$$\begin{aligned}\dot{z} &= Az + B\gamma(x)[u_f - \alpha(x)] \\ &= Az + B(-K_c\sqrt{p_5 - p_0})\left[G(u) - \frac{q_{bpp}}{K_c\sqrt{p_5 - p_0}}\right]\end{aligned} \tag{2.3}$$

in order to be feedback linerizable. Where $\gamma(x)$ is and must be nonsingular for all $x \in D$, and (A,B) controllable. Now that the requirements for feedback linearisation is stated, the rest of this section is going to focus on stating a linear system. To start the parameters is redefined in a simpler faction by denoting the parameters as.

$$a_j = \frac{\beta_j}{A_j l_j},\ b_j = \frac{A_j}{l_j \rho_j},\ c_j = \frac{K_{fric}}{l_j \rho_j},\ e_j = \frac{A_j g \Delta h_j}{l_j}$$

which forms the sate space model

$$\begin{aligned}\dot{p}_1 &= -a_1 q_1 - a_1 A_d v_d \\ \dot{p}_2 &= a_2 q_1 - a_2 q_2 \\ \dot{p}_3 &= a_3 q_2 - a_3 q_3 \\ \dot{p}_4 &= a_4 q_3 - a_4 q_4 \\ \dot{p}_5 &= a_5 q_4 - a_5 u_a \\ \dot{q}_i &= b_i p_i - b_i p_{i+1} - c_i q_i - e_i\end{aligned}$$

Where

$$u_a = q_{bpp} - K_c\sqrt{p_c - P_0}G(u)$$

Which is then written as a LTI system on state space matrix form as.

$$\dot{x} = \underbrace{\begin{bmatrix} 0 & -a_1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ b_1 & -c_1 & -b_1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & a_2 & 0 & -a_2 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & b_2 & -c_2 & -b_2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & a_3 & 0 & a_3 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & b_3 & -c_3 & -b_3 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & a_4 & 0 & -a_4 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & b_4 & -c_4 & -b_4 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & a_5 & 0 \end{bmatrix}}_{A} x + \underbrace{\begin{bmatrix} 0 & 0 \\ 0 & -e_1 \\ 0 & 0 \\ 0 & -e_2 \\ 0 & 0 \\ 0 & -e_3 \\ 0 & 0 \\ 0 & -e_4 \\ a_5 & 0 \end{bmatrix}}_{B} \begin{bmatrix} u_a \\ 1 \end{bmatrix} + \underbrace{\begin{bmatrix} -A_d a_1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix}}_{E} v_d$$

$$y = \underbrace{\begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}}_{C} x, \; x = \begin{bmatrix} p_1 & q_1 & p_2 & q_2 & p_3 & q_3 & p_4 & q_4 & p_5 \end{bmatrix}^\top$$

Definition (Hhalil, 1996) 1. *A nonelinear system*

$$\dot{x} = f_f(x) + G_f(x)u_f \tag{2.4}$$

where $f : D \to R^n$ and $G : D \to R^{n\times p}$ are sufficently smooth on s domain $D \subset R^n$, is said to be feedback linearisable (or input-state linearizable if there exists) a diffeomorphism $T : D \to R^n$ such that $D_z = T(D)$ contains the origin and the change of variables $z = T(x)$ transforms the system (2.4) into the form

$$\dot{z} = Az = B\gamma(x)[u - \alpha(x)] \tag{2.5}$$

with (A, B) controllable and $\gamma(x)$ nonsingular for all $x \in D$

2.6 Attempt to linearise with respect to depth

Model predictive control has become a success story in the cybernetics society. One of the reasons for the success of MPC is due to the ability to take account for physical constraints. Although the MPC produces great performance in control systems, it require lots of computation

power to produce the control action. Because there are many systems with fast dynamics that could gain a lot from the properties of MPC control, it's become more and more important to find solutions to speed up the computational time of a MPC. One way of speeding up the optimization is by changing the QP solver with an explicit solver, which gives birth to a range of new problems. One of these problems comes from the fact that the explicit MPC makes a dataset containing all the information needed to solve the QP. This dataset usually takes extremely long time to compute and if the system model changes at different depths, it becomes unrealistic to compute a new one at each depth. The solution to this problem is to create a liner state space model that doesn't change with the depth of the well.

First of we create the new states for the depth invariant state space model

$$x_i = \frac{1}{A_i} q_i \Rightarrow q_i = A_i x_i$$

$$p_i = z_i$$

The next step is to make a time transformation to cancel out the depth $\theta = t/\Delta h$ and presume that $l_i = \Delta h_i$, which makes way for the new derivatives.

$$\begin{aligned}
\frac{dx_i}{\partial\theta} &= \frac{1}{A_i}\frac{dq_i}{d\theta} \\
&= \frac{\Delta h}{A_i}\frac{dq_i}{dt} \Rightarrow \dot{q}_i = \frac{A_i}{\Delta h}\frac{dx_i}{d\theta} = \frac{A_i}{l_i}\frac{dx_i}{\partial\theta} \\
\frac{dp_i}{d\theta} &= \frac{dz_i}{dt}\Delta h \Rightarrow \dot{p}_i = \frac{1}{\Delta h}\frac{dz_i}{d\theta} = \frac{1}{l_i}\frac{dz_i}{d\theta}
\end{aligned}$$

Which gives the the new state space model

$$\begin{aligned}
\frac{1}{l_1}\frac{dz_1}{d\theta} &= \frac{\beta_1}{A_1 l_1}\left(-q_1 - v_d A_d\right) \\
&= \frac{\beta_1 l_1}{A_1 l_1}\left(-A_1 x_1 - v_d A_d\right) \\
&= -\beta_1 x_1 - \beta_1 \frac{A_d}{A_1} v_d \\
\frac{dz_1}{d\theta} &= -\beta_1 x_1 - \beta_1 \frac{A_d}{A_1} v_d \\
\frac{dz_2}{d\theta} &= \frac{\beta_2 A_1}{A_2} x_1 - \beta_2 x_2 \\
\frac{dz_3}{d\theta} &= \frac{\beta_3 A_2}{A_3} x_2 - \beta_3 x_3 \\
\frac{dz_4}{d\theta} &= \frac{\beta_4 A_3}{A_4} x_3 - \beta_4 x_4 \\
\frac{dz_5}{d\theta} &= \frac{\beta_5 A_4}{A_5} x_4 - \frac{\beta_5}{A_5} u_a \\
\dot{q}_i &= \frac{A_i}{l_i \rho_i}\left(p_i - p_{i+1}\right) - \frac{K_{fric}}{l_i \rho_i} q_i - A_i g \frac{\Delta h_i}{l_i} \\
\frac{A_i}{l_i}\frac{\partial x_i}{\partial \theta} &= \frac{A_i}{l_i \rho_i}\left(iz_i - iz_{i+1}\right) - \frac{K_{fric}}{l_i \rho_i}\frac{A_i}{1} x_i - A_i g \frac{\Delta h_i}{l_i} \\
\frac{\partial x_i}{\partial \theta} &= \frac{1}{\rho_i}\left(iz_i - iz_{i+1}\right) - \frac{K_{fric}}{\rho_i} x_i - g \Delta h_i
\end{aligned}$$

The drill string is presumed to have a radius r_{string} and the well is presumed to have a radius r_i in control volume i. Then the cross section area becomes

$$A_i = \pi(r_i^2 - r_{\text{string}}^2)$$

Where the drill string radius is presumed to be zero and therefore creating

$$\frac{A_i}{A_{i+1}} = \frac{r_i^2}{r_{i+1}^2} = \left(\frac{r_i}{r_{i+1}}\right)^2 \approx k_{\text{area}}$$

Which means because r_i and r_{i+1} are presumed linearly dependent, A_i and A_{i+1} becomes linearly dependent. We also presume that the drill bit area A_d and the cross section area of the lowest control volume is linearly dependent, and that the top cross section area A_5 doesn't change.

If all the presumptions above proves to hold then we have successfully created a depth invariant state space model. The fact of the matter is that most of these assumptions actually have a fighting chance, except for the very first one $\Delta h_i = l_i$. This is due to the fact that the well might be non-vertically as shown in figure 2.6. From this it can be concluded that although the depth of the well is included in the model, the model cannot be linearised with respect to depth because l_i and Δh_i may in general differ from each other.

To connect the dots regarding linearizing the system with respect to depth, this paragraph is intended as a summation of the main points above. First of the drilling model is none-linear and to even create a linear model for a specific depth the model needs to be linearized. This none-linearity is imposed by the choke valve and a linearization are presented in section 2.5. Although the problem now is linearized, it is still not time-invariant. In this section a method was presented to to create a model that does not change with depth in order to eliminate this time varying factor. This was partially successfully if the well is drilled just vertically. Which is of cause newer the case in a modern drilling operation. In other words the geometry of the well makes it practically impossible to create a time invariant model. From these discoveries it was, in agreement with supervisor, determined not to proceed with explicit MPC.

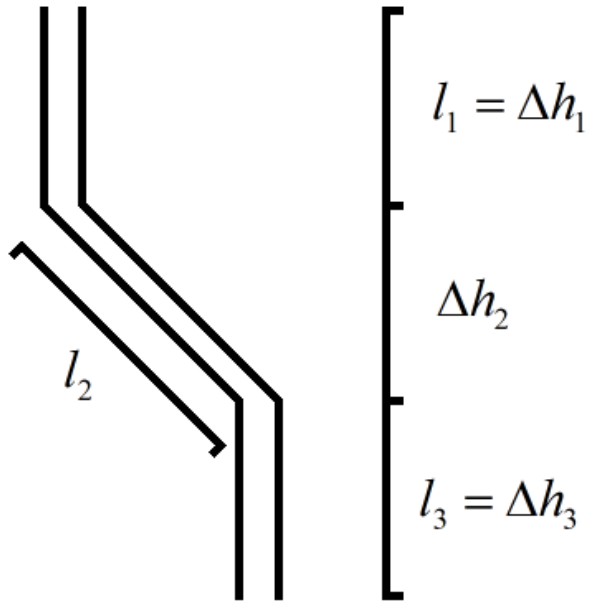


Figure 2.6: Non-vertical well

Chapter 3

Heave motion disturbance model and state estimation

3.1 Heave motion disturbance model

An important part of designing control systems is to evaluate the robustness in the presence of disturbances. In our case the disturbance is waves hitting the floating drill rig causing a heave motion, illustrated in figure 3.1.

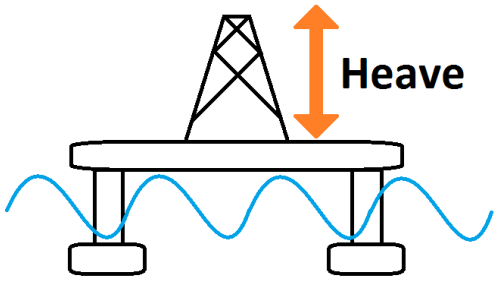


Figure 3.1: Direction of wave force on drilling rig

3.1.1 Wave spectrum

If wind blows over large areas of sea, ocean waves are born. And as the distance grow and the wind speed rises the waves grow taller. So when ship designers and control engineers want in-

formation about what waves are produced by a given wind in a specific area, they need a wave analysis. This wave analysis is often presented in form of a spectrum analysis. This spectrum may have more than one peak. This is due to the fact that tidal waves or existing waves often have a low frequency in opposition to newly formed waves with a higher frequency.

There has been a lot of research into this field and a lot of spectrum models have been created. But the relevant case is probably the Joint North Sea Wave Project (JONSWAP) spectrum (K. Hasselman, 1973). This spectrum was created as a joint venture between England, Holland, USA and Germany on the island of Sylt in the north of Germany. These measurements were taken by an array of sensors in recording for several weeks in 1968-1969. The spectral density function is written by (Fossen, 2011) as

$$S(j\omega) = 155\frac{H_S^2}{T_1}\omega^{-5}\exp\left(\frac{-944}{T_1^4}\omega^{-4}\right)\gamma^Y$$

where H_s is the significant wave height, T_1 the average wave period, $\gamma = 3.3$ and

$$Y = \exp\left[-\left(\frac{0.191\omega T_1 - 1}{\sqrt{2}\sigma}\right)^2\right]$$

where

$$\sigma = \begin{cases} 0.07 & \text{for } \omega \leqslant 5.24/T_1 \\ 0.09 & \text{for } \omega > 5.24/T_1 \end{cases}$$

3.1.2 Liner approximation

As (Fossen, 2011) describes a linear wave response approximation is usually preferred over a spectrum analysis by a control engineer due to it's simplicity and applicability. This linear response is presented in the form of a transfer function.

$$H(s) = \text{diag}\left[h^{\{1\}}, \ldots, h^{\{6\}}\right]$$

Were there are six transfer functions presented in a diagonal matrix. Because we only are interested in the heave response the transfer function can be presented as.

$$h^h(s) = \frac{K_\omega s}{s^2 + 2\gamma\omega_0 s + \omega_0^2}$$

where

$$K_\omega = 2\gamma\omega_0\sigma$$

In the paper (Amirhossein Nikoofard and Pavlov, 2014) it is stated that typical parameter choices for drilling operations in the north sea are

- $\lambda = 0.1017$
- $\sigma = 1.9528$
- $H_s = 4.7$
- $T_0 = 8.7$
- $\omega_0 = 0.7222$
- $K^h = 23$

Which will provide a linear spectrum describing the sea state as shown in fig 3.2.

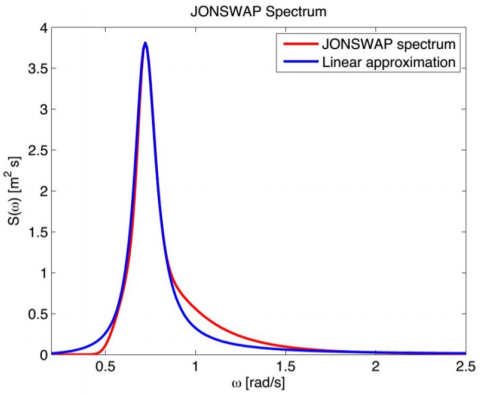


Figure 3.2: JONSWAP spectrum and its approximation. Copied from (Amirhossein Nikoofard and Pavlov, 2014)

3.1.3 Wave response

With the established wave spectrum of the north sea waves and the linear approximation model in hand, it is possible to simulate the wave amplitude. Knowing the waves is not enough to simulate the heave motion of a drilling rig. To transform the wave amplitude to heave motion, Response Amplitude operators (RAOs) are introduced. There are two main types of RAOs

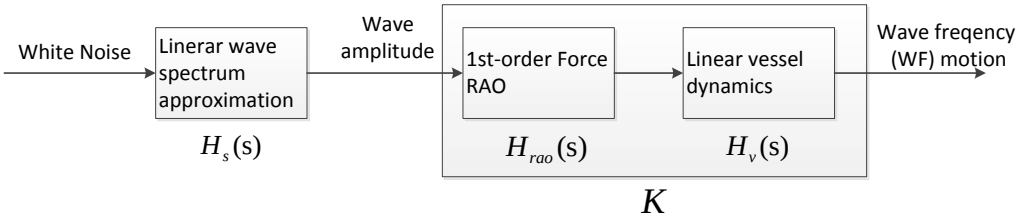


Figure 3.3: Linear approximation for computation of wave induced positions. Inspired by (Fossen, 2011)

- $H_{rao}(s)$ - **Wave force response amplitude operator**

  This is a transfer function that transforms the wave amplitude to a force in [N]. The function is creating a force vector by using newton's second law of motion.

- $H_v(s)$ - **Motion response amplitude operator**

  This transfer function transforms force to motion.

These functions are put together as shown in figure(fig). In addition to this the RAO vessel model can be approximated as.

$$H_{rao}(s)H_v(s) \approx K$$

Where K is a matrix of tunable gains.

$$K = \text{diag}\left[K^{\{1\}}, \ldots, K^{\{6\}}\right]$$

If the fixed gain approximation is applied, the generalized wave-frequency position vector becomes.

$$\eta_\omega = KH_s(s)\omega(s)$$

where $H_s(s)$ is the diagonal matrix containing all the linear approximations of the wave spectrum as described in the section above. Because we only are interested in the heave motion our wave-frequency position equation becomes.

$$\eta_\omega^h = K^h h_s^h(s)\omega^h(s)$$

where $h_s^h(s)$ is the spectral factor of the wave spectral density function $S(\omega)$ and $\omega^h(s)$ is a zero-mean Gaussian white noise process with unity power across the spectrum:

$$P_{\omega\omega}^{dof}(\omega) = 1$$

This calculation will provide a JONSWAP wave approximation that generates waves like the ones in figure 3.4.

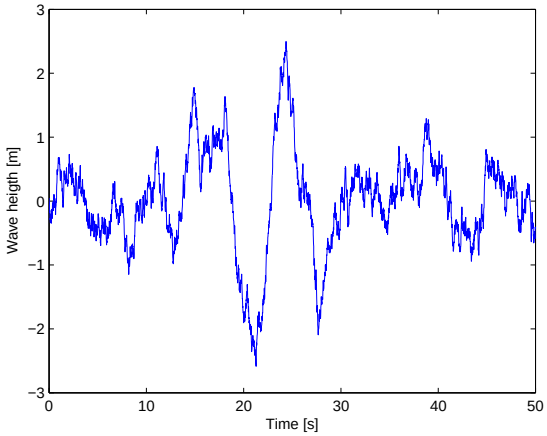


Figure 3.4: Waves generated from JONSWAP linear approximation

3.2 Kalman filtering

Kalman filtering is a type of state estimator created by Rudolf E. Kálmán. This technology uses a series of noisy measurements observed over time to produce estimates of unknown variables.

The Kalman filter itself is an efficient predictor-corrector algorithm that calculates a Kalman gain K_k in a way that minimizes the estimated error covariance $P_k^- = E[e_k^- e_k^{-\top}]$ where $e_k = x_k - \hat{x}_k^-$. The filter is commonly used on linear stochastic systems. Which means that the system has process noise and or measurement noise.

For more information and complete derivation of the Kalman filter see (R.G. Brown, 2012).

3.2.1 Design

A linear dynamical system can be described on explicit discrete steady state form as

$$x(k+1) = A_d x(k) + D_b u(k) + E_d v_d(k)$$
$$y(k) = A_d x(k)$$

Where the disturbance that affects the system can be modeled as.

$$w(k) = E_d v_d(k)$$

Where v_d represents the wave velocity on discrete form, and Ed is the discrete disturbance input matrix.

With information about the disturbance it is possible to create a covariance matrix.

$$E[w_k w_i] = \begin{cases} G, & i = k \\ 0, & i \neq k \end{cases}$$

And because we only are interested in the diagonal and $E[w_k w_k] = E[w_k^2]$ The covariance matrix becomes.

$$G = \text{diag}\left(E[w_1^2], E[w_2^2], \ldots, E[w_n^2]\right)$$

Now we have all the information we need to calculate the estimated states $\hat{x}_k$ using the Kalman filter algorithm presented in (R.G. Brown, 2012).

1. Enter prior estimate $\hat{x}_0^-$ and it's error covariance P_0^-
2. Compute calman gain:
   $K_k = P_k^- C_d^\top \left(C_d P_k^- C_k^\top\right)^{-1}$
3. Update estimate with measurment z_k:
   $\hat{x}_k = \hat{x}_k^- + K_k \left(y_k - C_d \hat{x}_k^-\right)$
4. Compute error covariance for updated estimate:
   $P_k = (I - K_k C_d) P_k^-$
5. Project ahead:
   $\hat{x}_{k+1}^- = A_d \hat{x} + B d u$
   $P_{k+1}^- = A_d P A_d^\top + G$
6. Increase $k = k + 1$ and go to step two.

3.2.2 Stability

Stability theorem

$$x(k+1) = A_d x(k) + D_b u(k) + w(k)$$
$$y(k) = A_d x(k) + v(k)$$

If the system above is a time-invariant, observable and statistically reachable. And G is positive definite. Then the following statements are true for the Kalman filter.

- For any symmetric and positive definite matrix $P(0|0)$, $P(k|k)$ converges uniformly to a unique matrix $\bar{P}$. Which implies that $K(k)$ converges to a constant matrix $\bar{K}$.

- The steady state Kalman filter is defined as the one using the Kalman gain $\bar{K}$. We have $\hat{x}(k|k) = \hat{x}(k|k-1) + \bar{K}(z(k) - H\hat{x}(k-1)) = (\Phi - \bar{K}H\Phi)\hat{x}(k-1|k-1) + \bar{K}z(k)$. This filter is asymptomatically stable, i.e. all the eigenvalues of $(\Phi - \bar{K}H\Phi)$ lie within the unit circle.

This is important because they provide the necessary condition for the error covariance matrices to converge independently of the initial condition P(0|0), and guarantee numerical stability of the filter equations. (Erik Bølviken, 1998)

- The system is unstable or has a bias error.
- The model error is higher than expected

Chapter 4

Model Predictive Control

4.1 Feasibility

An equation such as $2x - 6 = 0$ has only one solution or feasible position $x = 3$, a second degree equation such as $x^2 - 4 = 0$ has two feasible positions $x = -2 \vee x = 2$. The multi variable equation $z_1^2 + z_2^2 = 0$ has infinitely many feasible points forming a feasible region, where all the points form a circle with radius one. One might reduce this feasible region by adding a new constraint $z_1 + z_2 \geqslant 0$, which forms a smaller feasible region shown in red on figure 4.1. These constraints form what is called a feasible set.

$$\begin{aligned}\Omega &= \left\{z \in \mathbb{R}^2 \mid c_1(z) = 0 \wedge c_2(z) \geqslant 0\right\} \\ &= \left\{z \in \mathbb{R}^2 \mid z_1^2 + z_2^2 = 0 \wedge z_1 + z_2 \geqslant 0\right\}\end{aligned}$$

Which separates constraints in to groups, equality constraints $\varepsilon = \{1\}$ where one can find $c_1(z)$, and inequality constraints $\mathscr{I} = \{2\}$ where we find constraint $c_2(z)$.

4.2 Optimization

In examples such as above where there might be infinitely many solutions one stops searching for a solution, and starts searching for the optimal solution inside the given constraints. This

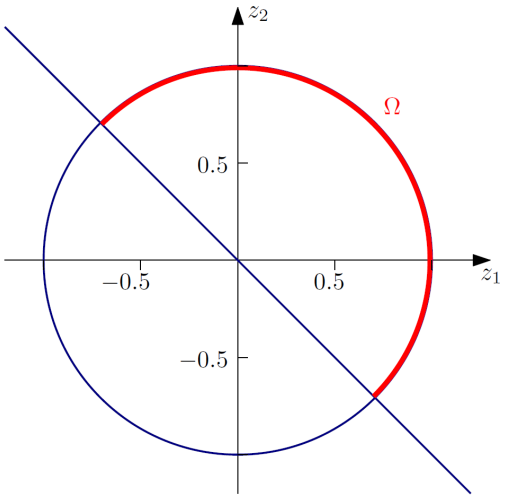


Figure 4.1: Ilustration of feasible region. Copied from (Heirung, 2013)

form of thoughts form the optimization problem.

$$\min_{x \in \mathbb{R}^n} f(x)$$

Subject to

$$c_i(x) = 0, \quad i \in \varepsilon$$
$$c_i(x) \geqslant 0, \quad e \in \mathscr{I}$$

The optimization problem has three main components, the decision variables $x = [x_1, x_2, \ldots, x_n]^\top$, an objective function $f(x)$ and the constraints $c_i(x)$. Where the objective function finds the optimal value for each decision variable within the given constraints. This is supposed to summarize the main points of the active-set method with the intent to give some clarity of the concept of how iterative constraint optimization work.

4.2.1 Constrained optimization

This section will describe the iterative active set method for solving a quadratic optimization problem, where G is positive semidefinite. Which means that $q(x)$ is a convex function and by theorem 1 the local minimizer x^* found by the the optimization is a global minimizer. The QP

problem is defined as

$$\min_x \quad q(x) = \tfrac{1}{2}x^\top Gx + x^\top c \quad \text{Subject to} \quad a_i^\top x = b_i, \quad i \in \mathscr{A}(x^*) \tag{4.1}$$

Where $\mathscr{A}(x^*)$ is the active set from definition 1 at the minimal point x^*. And usually there is no prior knowledge of the minimal point x^*

Line search

One of the most efficient ways of finding the minimum of a constrained optimization problem is by using a iterative search method. In each iteration of a line search, a direction of the search is computed. The iteration is given by.

$$x_{k+1} = x_k + \alpha_k p_k \tag{4.2}$$

In order for the line search to be efficient and successful the direction p_k and the step length α_k must be carefully chosen.

In each iteration the current set being worked on is denoted as the kth iterate x_k by $\mathscr{W}_k$. For each iteration it is required that the gradients a_i of the working set $\mathscr{W}_k$ are linearly independent. Also a check to see if x_k and $\mathscr{W}_k$ minimizes (4.1) is done in each iteration. The sub problem for each iteration is defined as

$$p = x - x_k, \quad g_k = Gx_k + c$$

And by substituting for x into the objective function in (4.1), we get a new objective function

$$q(x) = q(x_k + p) = \frac{1}{2}p^\top Gp + g_k^\top p + q(x_k)$$

Here, $q(x_k)$ is independent of p and can be dropped from the objective function without changing the solution of the problem. Which results in the QP problem to be solved for each iteration.

$$\begin{aligned} \min_p \quad & \tfrac{1}{2}p^\top Gp + g_k^\top p \\ \text{Subjet to} \quad & a_i^\top p = 0, \quad i \in \mathscr{W}_k. \end{aligned} \tag{4.3}$$

Where the solution of optimization problem (4.3) is the search direction p_k in (4.2). Since G is positive definite, the solution of (4.3) can be solved as a direct solution of the KKT system.

To maximize the decrease in q, the step length α_k is chosen to be as long as possible in $[0,1]$ while retaining feasibility by the following definition.

$$\alpha_k \triangleq \min\left(1, \min_{i \notin \mathcal{W}_k, a_i^\top p_k < 0} \frac{b_i - a_i^\top x_k}{a_i^\top p_k}\right) \tag{4.4}$$

This way of iterating is used until $P_k = 0$ then we have that

$$\sum_{i \in \hat{\mathcal{W}}} a_i \hat{\lambda}_i = g = G\hat{x} + c \tag{4.5}$$

This is supposed to summarize the main points of the active-set method with the intent to give some clarity of the concept of how iterative constraint optimization work. For a more detailed dive into the active set method or optimization in general read (Wrigth, 2006). It also gives the opportunity to present an example of the active set algorithm from presented in algorithm 1, as

a example how to solve a convex QP.

Compute a feasible starting point x_0;

Set $\mathcal{W}_0$ to be a subset of the active constraint at x_0;

for $k = 0, 1, 2 \ldots$ **do**

  Solve (4.3) to find p_k ;

  if $p_k = 0$ **then**

    Compute Lagrange multipliers $\hat{\lambda}_i$ that satisfy 4.5, with $\hat{\mathcal{W}} = \mathcal{W}_k$;

    if $\hat{\lambda}_i \geq 0 \ \forall \ i \in \mathcal{W}_k \cap \mathcal{I}$ **then**

      Stop with solution $x^* = x_k$;

    else

      $j \leftarrow \arg\min_{j \in \mathcal{W}_k \cap \mathcal{I}} \ \hat{\lambda}_j$;

      $x_{k+1} \leftarrow x_k$;

      $\mathcal{W}_{k+1} \leftarrow \mathcal{W}_k \setminus \{j\}$;

    end

  else

    Compute α_k from (4.4) ;

    $x_{k+1} \leftarrow x_k + \alpha_k p_k$;

    if *There are blocking constraints* **then**

      Obtain $\mathcal{W}$ by adding one of the blocking constraints to $\mathcal{W}_k$;

    else

      $\mathcal{W}_k \leftarrow \mathcal{W}_k$

    end

  end

end

Algorithm 1: Active-set method for Convex QP. Copied from (Wrigth, 2006)

Definition (Wrigth, 2006) 1.

The active set $\mathcal{A}(x)$ at any feasible c consists of the equality constraint indices from ε together with the indices of the inequality constraints i for which $c_i(x) = 0$; that is,

$$\mathcal{A}(x) = \varepsilon \cup \{i \in \mathcal{I} | c_i(x) = 0\}.$$

At a feasible point x, the inequality constraints $i \in \mathscr{I}$ is said to be active if $c_i(x) = 0$ and inactive is the strict inequality $c_i(x) > 0$ is satisfied.

Theorem (Wrigth, 2006) 1.
When f is convex, any local minimizer x^* is a global minimizer of f. If in addition f is differential, then any stationary point x^* is a global minimizer of f.

4.2.2 Dynamic Optimization

Ordinary mathematical systems can be divided into static or dynamical systems, where dynamical systems can be divided into linear and non-linear systems.

- **Statical systems**
  Statical systems are time independent systems, or systems that is not a function of time. To put it in a practical perspective, to apples plus thee apples always equals five apples.

- **Dynamical systems**
  This is not necessary a system that changes over time, but is a function of time. To put this into a practical perspective let's say you have a car that drives around in $14[\mathrm{m/s}]$ and have an acceleration of $1[\mathrm{m/s}]$, then it is not going in $14[\mathrm{m/s}]$ five minutes later. The modern idea of dynamics comes from Newtonian science and the tree laws of motion but there are dynamics in everything, ranging from social dynamics to economical dynamics, one only need to look at a border picture and its there.

Until now the main theme have bean optimization of statical systems or time independent systems, but from now on we are going to look at dynamical systems. More specifically linear dynamical systems written on state space form.

$$x_{t+1} = Ax_t + Bu_t + Ev_t$$

Where the system variables are

- x - System states
- u - System input

- v - System disturbance

where the first two are the decision variables. As one can imagine these systems does not have one answer, but an array where the length depends on the sampling time and the length of the control horizon. This introduces two new concepts to be described in more detail, sampling time and control horizon.

Control horizon

Optimizing a dynamical system is not about optimizing a set of variables but a set of variables over time as shown in the upper figure in figure 4.2. This means one need to have a decision variable for each state and control input at every discrete time instant. If the control horizon is infinitely long there will be infinitely many decision variables and it will take infinitely long time to get an answer. So how long should the horizon be? "The prediction horizon is commonly chosen sufficiently long for the plant to reach steady state"(Hovd, 2012). This is because of stability issues. Lets say that the plant undershoots early in the step response, then the optimization would automatically give a negative response to compensate if the control horizon wasn't sufficiently long.

Sampling time

Both the length of the control horizon and the sampling time are important issues because they affect the complexity of the optimization problem. This doesn't mean that the sampling rate can be chosen to be as small as possible, because the system must have a decent sampling frequency for the optimization to be efficient, and the systems performance gets to some extent better as the frequency rises depending on how quick the system dynamics are. There is also the aspect of disturbance rejection which also improves as the frequency rises. The Nyquist–Shannon sampling theorem (Marks, 1991) states that.

$$f_s > 2f$$

Where f_s is the sampling frequency and f the highest frequency component of interest. This means that to be able to remove a disturbance, the sampling frequency has to be at least twice as high as the disturbance you want to remove. In our case the waves has it's main energy focused

between $0.5 - 1.5$[rad/s] with a peak at $\omega = 0.722$[rad/s]. Due to the fact that 1.5[rad/s] ≈ 0.24Hz the sampling theorem states

$$f_s > 2f \approx 2 \cdot 0.24 = 0.48$$
$$T_s = \frac{1}{f_s} \approx 2.1$$

Which means the sampling time must at least be smaller than $T_s = 2.1$ for the disturbance rejection to be efficient.

Dynamical optimization problem formulation

The typical formulation of a linear optimization problem used in most research literature is formulated in section 4.4, and the formulation used in ACADO toolbox for generating a MPC solver is

$$\min_{z \in \mathbb{R}^n} \int_{t_0}^{t_0+T} \left\| h(t, x(t), u(t), p) - \eta(t) \right\|_Q^2 dt + \left\| m(x(t_0+T), p, t_0+T) - \mu \right\|_P^2$$

Subject to:

$$\begin{aligned} & x(t_0) = x_0 \\ \forall\, t \in [t_o, t_o+T]: \quad & 0 = f(t, x(t), \dot{x}(t), u(t), p) \\ \forall\, t \in [t_o, t_o+T]: \quad & 0 \geqslant s(t, x(t), u(t), p) \\ & 0 = r(x(t_0+T), p, t_0+T) \end{aligned}$$

Where

- x - The states
- u - The control input
- p - A time-constant parameter
- T - The time horizon
- f - Represents the model equations

- s - The path constraints
- r - The terminal constraints.

This is of course a non-linear set-up because ACADO is a non-linear optimization library, but that does not mean it cannot be used on linear problems, in fact the toolbox identifies the system as linear and creates a linear QP solver. It's also worth mentioning that the dynamical optimization problems are formulated continuously and automatically discretized by the toolbox. The implementation will be covered in more detailed in chapter 6.

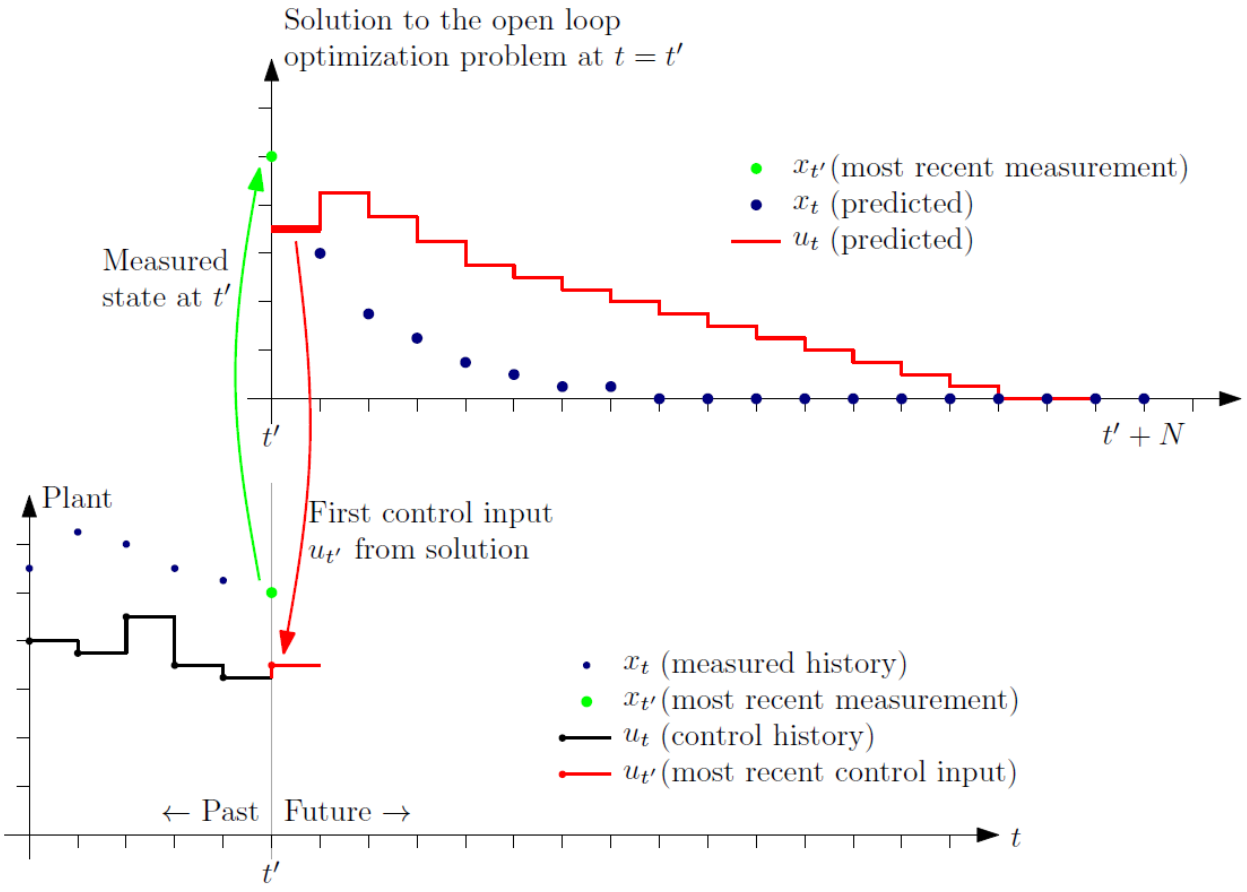


Figure 4.2: Illustration of the MPC principle. Copied from (Heirung, 2013)

4.3 Optimal control

Optimization without feedback is great for solving minimizing/maximizing problems, but using this approach on a dynamical system will only generate an optimal path N steps into the future. This would have worked if the system model was perfectly modelled, without disturbances and the last output was to stay the same for all foreseeable future. Consequently the idea of optimization without feedback becomes less plausible and in most cases only a hypothetical idea. Which gives birth to optimization with feedback, often referred to as *Model Predictive Control*(MPC), but also referred to as *Receding Horizon control* and *Moving Horizon Optimal Control*. This concept is implemented by solving a optimization problem for each sampling instance as described by (D. Q. Mayne, 2000).

> *"Model predictive control is a form of control in which the current control action is obtained by solving, at each sampling instant, a finite horizon open loop optimal control problem, using the current state of the plant as the initial state; the optimization yields an optimal control sequence and the first control in this sequence is applied to the plant."*
>
> —D. Q. Mayne (2000)

The functionality of the MPC can be seen in figure 4.2. In the figure an optimization problem is solved for a given time instance with the initial condition x_0 (upper figure), sets the first control input to the proses(lower figure), let the system iterate one sampling instant into the future before reading out the state information from the process and feed it into the optimization problem and solves it for the next time instant. The MPC functionality can then be compressed into the short algorithm 2.

for $t = 0, 1, 2 \ldots$ **do**

Get the current state x_t;

Solve a dynamic optimization problem on the prediction horizon from t to $t + N$ with x_t as the initial condition;

Apply the first control move u_t from the solution above;

end

Algorithm 2: State feedback MPC procedure. Copied from [optcontrol note]

4.4 Optimality and Stability

In this section there will be presented presented some specific ways of ensuring stability in MPC controllers. There will also be explained how to ensure feasibility in the controller at al times. In research literature MPCs is almost always presented presented on discrete state space form.

$$x(t+1) = Ax(t) + Bu(t), \quad x(0) = x_0 \tag{4.6a}$$

$$u(t) = Cx(t) \tag{4.6b}$$

Where $x(t) \in \mathbb{R}^n$ denotes the states at time t, $u(t) \in \mathbb{R}^m$ the input, and $y(t) \in \mathbb{R}^p$ the output. By iterating the discrete system in 4.6, k steps into the future, the states can be denoted as $x(t+k|t)$. And the optimization problem used in the MPC algorithm defined in algorithm 2, is defined as.

$$\begin{aligned} \min_{u,x} \quad J(u,x) = & \sum_{k=0}^{N_p-1} x^\top(t+k|t)Qx(t+k|t) \\ & + \sum_{k=0}^{N_m-1} u^\top(t+k|t)Ru(t+k|t) + x(N_p)^\top P_0 x(N_p) \end{aligned} \tag{4.7a}$$

subject to

$$F_1 u(t+k|t) \leq G_1 \tag{4.7b}$$

$$E_1 x(t+k|t) \leq G_1 + F_2 u(t+k|t) \leq G_2 \tag{4.7c}$$

and

$$\text{Stability Constraints} \tag{4.7d}$$

Where N_p is the length of the prediction horizon in samples, N_m is the length of the input horizon and $N_m \leq N_p$. An infinite horizon is defined as $N_p = \infty$, and finite horizon as $N_p =$ scalar.

Assumption (Morari, 1999) 1. *The polyhedron* $\{(x,u) : F_1 u \leq G_1 E_2 x + F_2 u \leq G_2\}$ *contains the origin* $(x = 0, u = 0)$. *And that the constraint* (4.7d) *are inserted in the optimization problem are implemented in the optimization problem in order to guarantee closed loop stability.*

4.4.1 Defining stability constraints

A range of different techniques are used in literature to enforce stability assuming assumption 1 is satisfied (Morari, 1999). These approaches are divided into two main classes. The first uses the Lyapunov function $V(t) = J(u^*, x^*, N_p, N_m)$. Where u^* and x^* are the optimal solution from the optimization at each sampling instance. The second requires that $x(t)$ is shrinking in some norm. Some of the methods for guaranteed stability are listed below.

- **Terminal Constraint**

  One way of ensuring stability is to replace equation (4.7d) with the terminal constraint method defined as

  $$x(t + N_p|t) = 0 \tag{4.8}$$

  The main drawback to this approach is that all the states have to be brought to zero within the prediction horizon. This might require a large control effort in order to get the state to zero within the control horizon. The large control effort might also become a feasibility problem.

- **Invariant Terminal Constraint**

  The idea of the invariant terminal constraint is to relax the terminal constraint 4.8 to

  $$x(t + N_p|t) \in \Omega$$

  and set $u(t + k|t) = F_{LQ}x(t + k|t),\ \ k \geq N_m$ where F_{LQ} feedback gain. The set Ω is invariant under LQ control and such that the constraints are fulfilled inside the feasible set Ω.

- **Infinite Output Prediction Horizon** The constraint in (4.7d) is not required if $N_p = \infty$ and the system in (4.6) is asymptotically stable.

4.4.2 Feasibility

Feasibility of the optimization problem in (4.7a) must be ensured for each sampling instant for the system in (4.6), and if the system is feasible at $t = 0$ it can be assumed to be feasible for the rest of the control horizon. Feasibility is secured if the constraints are never broken (Morari,

1999). The constraints are usually divided into two types, hard constraints and soft constraints. Hard constraints can never be broken while soft constraints can be broken to ensure feasibility. Input constraints are usually constrained physically and can not be broken. The other constraints are the state constraints imposed by the fracture pressure and the formation pressure presented in section 2.1. These constraints can be broken under extreme circumstances to ensure feasibility. They can therefore be added as soft constraints. The most common way of adding soft constraints are by adding slack variables to the hard constraints, as shown below.

$$\epsilon x_{\text{min,k}} \leq x_k \leq \epsilon x_{\text{max,k}}$$

Where the slack variable ϵ is then added to the cost function with high cost.

4.5 Numerical integrator

Simulations are an important part of engineering a control system. It's a useful tool throughout the life cycle of a control system, beginning in the design but also in both maintenance and upgrading. Most control systems are dynamical and therefore formulated with either differential equations or transfer functions with initial conditions. Because computers are digital, simulating these systems requires numerical integrators. Simulations are used to simulate dynamical systems for implementation of controllers, designing controllers and testing the controllers before implementing them in a real system. In NMPCs it's often used as a tool for simulating the slope of the optimization problem in question. The reason why this is brought up is that it is used in the ACADO toolbox for creating MPC solvers.

Explisit Runge-Kutta

There are a bunch of integrator schemes and they are often placed into two categories, implicit and explicit methods. As the names implies, the explicit methods are easier to solve because they can be solved directly. The stability properties of implicit methods are usually to some extent better and therefore used when they are necessary. An example where implicit integrators are needed are stiff systems. The explicit numerical scheme for simulating an ODE on the form $\dot{y} = f(y,t)$ with an explicit Runge-kutta method with σ stages is given by.

$$k_i = f(y_n + h\sum_{j=1}^{i-1} a_{ij}k_j, t_n c_i h), \quad i = 1,\dots,\sigma \tag{4.9a}$$

$$y_{n+1} = y_n + h\sum_{j=1}^{\sigma} b_j k_j \tag{4.9b}$$

One of the most common explicit Runge-Kutta methods is the forth order model often shortened RK4. The butcher tableau which gives the constants for the numerical scheme in (4.9) is

$$
\begin{array}{c|cccc}
0 & & & & \\
c_2 & a_{21} & & & \\
c_3 & a_{31} & a_{32} & & \\
\vdots & \vdots & \vdots & \ddots & \\
c_\sigma & a_{\sigma 1} & a_{\sigma 2} & \cdots & a_{\sigma,\sigma-1} \\
\hline
 & b_1 & b_2 & \cdots & b_{\sigma-1} \quad b_\sigma
\end{array}
=
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{2}{6} & \frac{2}{6} & \frac{1}{6}
\end{array}
$$

For more information about numerical integrators, simulation of ODEs or information about stability of numerical integration schemes, please refer to the book (Olav Egeland, 2002).

4.6 Condensation

Model predictive control has shown itself to be a powerful way of controlling dynamical systems. This is both because it produces great results and have a native property of handling constraints. Although the MPC produces great performance in control systems it has a price, it require lots of computation power to solve the optimization problem for each sampling instance to produce the control action. Because there are many systems with fast dynamics that could gain a lot from the properties of MPC control, it's become increasingly important to find solutions to speed up the computational time of a MPC. One of the ways to minimize the computational time is by minimizing the number of decision variables. In literature there are a range of different ways to minimize the number of decision variables. But when boiling down these different approaches there are mainly two extremal approaches, **sparse formulation** where both states and inputs are decision variables and **condensed formulation** where only inputs are decision variables. There are also approaches in between often called **sparse-condensed formulations**.

- **Sparse formulation**

  In the sparse formulation both the states and the inputs are decision variables. Which forms the decision variable vector for a QP problems as.

$$z = \begin{bmatrix} x_0^\top & u_0^\top & x_1^\top & u_1^\top & \cdots & u_{N-1}^\top & x_N^\top \end{bmatrix}^\top$$

  To the corresponding QP problem

$$\begin{aligned} \min_z \quad & J(z) = \tfrac{1}{2} z^\top H z \\ \text{subject to} \quad & Fz = f \\ & Gz \leq g \end{aligned}$$

- **Condensed formulation**

  In the Condensed formulation only the inputs are used as decision variables. Which forms the decision variable vector for a QP problems as.

$$u = \begin{bmatrix} u_0^\top & u_1^\top & u_2^\top & \cdots & u_{N-1}^\top \end{bmatrix}^\top$$

  To the corresponding QP problem

	Computation Time	Memory consumption
Condensed	$\mathcal{O}(N^3 m^2 (l+m))$	$\mathcal{O}(N^2 m(l+m))$
Sparse	$\mathcal{O}(N(m+n)^2 (l+m+n))$	$\mathcal{O}(N(m+n)(l+m+n))$

Where

Variable	Definition
m	Number of inputs
n	Number of states
l	Number of constraints
N	Length of horizon in samples

Table 4.1: Computational complexity and memory requirements from (Juan L. Jerez, 2011)

$$\begin{aligned} \min_{u} \quad & J(u) = u^\top H u + (F\theta + f)u \\ \text{subject to} \quad & A_{ieq} u \leq b_{ieq} + B_{ieq}\theta \end{aligned}$$

Which comes from the condensed formulation used in the rest of this thesis, for the complete formulating look in appendix A.

From this it might seem like the condensed solution is always the way to go but this is neither true or tangible. To get something more tangible, one should look at table 4.1, to see what solution to choose. In general the condensed solution is the better choice for both memory consumption and computation time if N is small. If N is large, a sparse formulation is probably the way to go. How long the horizon can be before the sparse solution becomes sensible is dependent on the number of states, inputs and constraints. Fore more information on computation time and memory consumption in condensation read (Juan L. Jerez, 2011).

Chapter 5

Controller design

5.1 Control hierarchy

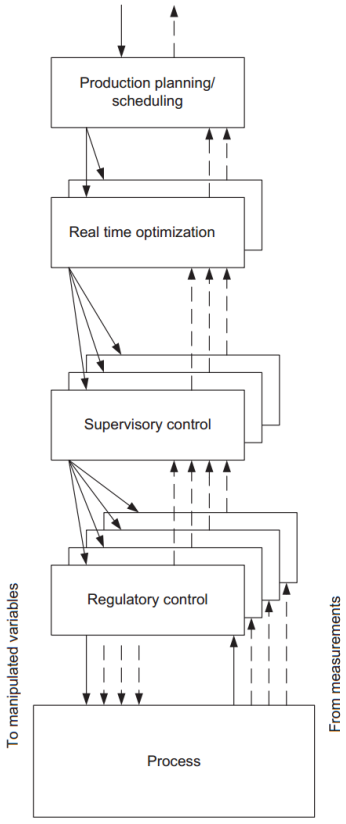


Figure 5.1: Typical structure of the control system for a large plant in proses industry. Copied from (Hovd, 2012)

A typical structure of a modern control system is show in figure 5.1. Beginning at the bottom of this control system structure one finds the process layer. This layer is literally the process as it is in the drilling hydraulic model .

Above this layer is some connections, these connections represents measurements, in this case the top side pressure. The outputs in this case is the choke set-point. The next level is the regulatory control layer. This is where the low level controller is. In this case, this it is the controller that controls the topside pressure with the choke valve. This layer is in ordinary cases important because it's stabilizes the system for easier MPC control. We do not use a PID controller to stabilise the system, but this can be done and is a common approach to stabilize the system. This of course modifies the dynamical state space model of the plant, and a new state space model must be used in the MPC controller. An example of how to modify the system state space model to contain the PI controller is attached in appendix B. In this implementation this control layer consists of a feedback Linearization to remove the non-linearity in the state space model.

The next level is the Supervisory level. This is where the MPC is placed and it's using the estimated states from the Kalman filter to calculate an optimal manipulated value as a choke set-point. This way one can indirectly control the drill bit pressure with the MPC controller.

The next layer is the real time optimization(RTO) control layer. In this layer the optimal conditions to the MPC is set. This means setting the pressure limits for the drill bit pressure between the limits described in section 2.1. And setting the set point. These values changes at different depths and different materials, and these values are calculated before the drilling starts by a geologist.

The last layer is essentially exactly what the name indicates.

5.2 System properties

5.2.1 Simulation Parameters

To simulate the MPD system in section 2.3. The well is assumed to be 1990.99 [m] long and the identified parameters from the IRIS Drill simulator are used (Amirhossein Nikoofard and Pavlov, 2014). Where the parameters are defined as.

Parameter	Value	Parameter	Value
a_i	$2.545 \cdot 10^8$ $[Pa/m^3]$	K_f	$5.725 \cdot 10^5$ $[aPa/m^3]$
b_i	$5.725 \cdot 10^{-8}$ $[m^4/Kg]$	K_g	0.00650
c_i	14.4982 $[1/sm^2]$	A	0.0269 $[m^2]$
e_i	0.2638 $[m^3/s]$	A_d	0.0291 $[m^2]$
g	9.806 $[m/s^2]$	K_c	2.32 $[m/s^2]$
p_0	101325 $[Pa]$	q_{bpp}	369.2464 $[m^3/s]$

(5.1)

5.2.2 Choke Valve Characteristic

In section 2.4.2 a choke valve model was presented. In this section the choke valve characteristics was not presented. To simulate the systems choke valve, parameters identified from data from a offshore drilling rig is used. The identification was done by the author in a previous course in system identification on data presented in (Kaasa, 2012). This choke characteristic is denoted by by the parameters in table 5.1 The differential pressure over the valve is formulated as $p_5 - p_0 = \Delta p$. In order to create the simulated choke characteristic in figure 5.2 the differential pressure is assumed to be $\Delta p = 25[bar]$. From this figure one can clearly see the non-linearity imposed by the $G(u)$. The choke actually doesn't start opening before it has moved approximately 35 percent of the range, but after that it seams to have a quick-opening characteristic.

Variable	Value
K_c	8.9670
a_g	$-1.96 \cdot 10^{-4}$
b_g	0.0419
c_g	-1.1920

(5.2)

Table 5.1: Choke characteristic

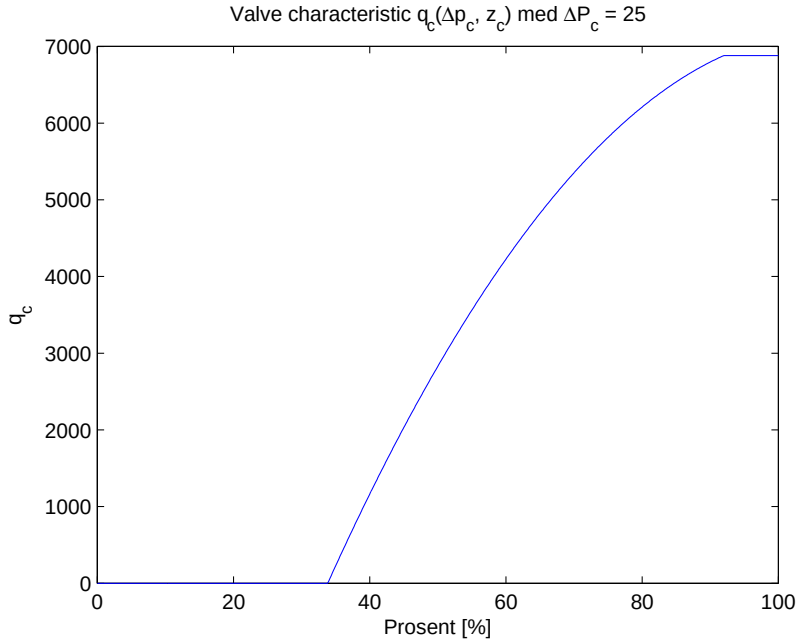


Figure 5.2:

5.2.3 Controllability

The controllability matrix C, from theorem 1 has the row rank.

$$\text{rank}(C) = 9$$

Which means the liner system (A,B) from section 2.5 is controllable with the parameters from (5.1). Since the system is stabilizable, which is a weaker form of controllable, the system in (5.9) will always have a well defined solution. It also means that that the feedback linearisation from section 2.5 is satisfied by definition 1.

Theorem 1. *(Chen, 2009)*

The n-dimensional pair (A, B) *is controllable if the* $n \times np$ *the controllability matrix*

$$C = \begin{bmatrix} B & AB & A^2B \cdots A^{n-1}B \end{bmatrix}$$

has rank n (full row rank)

5.2.4 Observability

The observability matrix O, from theorem 1 has the column rank.

$$\text{rank}(O) = 9$$

Which means the liner system (A,C) from section 2.5 is observable with the parameters from (5.1). Because the system is observable and therefore automatically stochastically reachable by [ref oslo]. The Kalman filter in section 3.2 then satisfies the stability theorem in section 3.2.2.

Theorem 1. *(Chen, 2009)*

The n-dimensional pair (A, C) *is observable if the* $np \times n$ *the Observability matrix*

$$O = \begin{bmatrix} C \\ CA \\ \vdots \\ CA^{n-1} \end{bmatrix}$$

has rank n (full column rank)

5.2.5 Internal Stability

The MPD system from section 2.5 with the parameters from (5.1) on state space form with the parameters denoted (5.1) with the characteristic polynomial

$$\det(\mathbf{A} - \lambda \mathbf{I}) = 0$$

Where the roots are.

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9
-14.2397	-13.5127	-12.4762	-11.4547	-3.0435	-2.0220	-0.0000	-0.2585	-0.9855

Which is satisfying theorem 1.2 and the system is marginally stable. Because the system only is marginally stable the infinite output prediction horizon stability concept can not be used. To guarantee MPC stability the solution then becomes to use the invariant terminal constraint to ensure stability. More information about MPC stability can be found in section 4.4.

Theorem 1. *(Chen, 2009)*

1. *The equation* $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t)$ *is marginally stable if and only if all the eigenvalues of* $\mathbf{A}$ *have zero or negative real part and those with zero real parts are are simple roots of the minimal polynom of* $\mathbf{A}$

2. *The equation* $\dot{\mathbf{x}}(t) = \mathbf{A}\mathbf{x}(t)$ *is asymptotically stable if and only if all the eigenvalues of* $\mathbf{A}$ *have negative real part.*

5.3 System discretization

When modelling a dynamical system it's ordinary to look at the plant as a continuous system, because physics in nature usually are continuous. To use these systems in control applications they are usually denoted on a standard form suitable for the system. One of the most common formulation for LTI system is the matrix state-space form.

$$\dot{\mathbf{x}}(t) = \mathbf{A}x(t) + \mathbf{B}\mathbf{u}(t) \tag{5.3a}$$

$$\mathbf{y}(t) = \mathbf{C}\mathbf{x}(t) + \mathbf{D}\mathbf{u}(t) \tag{5.3b}$$

Modern computers on the other hand are digital and aren't made to handle continuous dynamics directly. In order to implement these systems on modern computers these systems usu-

ally are discretized. A discrete state space model can be denoted on the form.

$$\mathbf{x}[k+1] = \mathbf{A_d}\mathbf{x}[k] + \mathbf{B_d}\mathbf{u}[k] \tag{5.4a}$$

$$\mathbf{y}[k] = \mathbf{C_d}\mathbf{x}[k] + \mathbf{D_d}\mathbf{u}[k] \tag{5.4b}$$

Where the system matrices on discrete form is denoted on the following form where T_s is the sampling time.

$$\mathbf{A_d} = e^{AT_s}, \quad \mathbf{B_d} = \left(\int_0^T e^{\mathbf{A}\tau} d\tau\right)\mathbf{B}, \quad \mathbf{C_d} = \mathbf{C}, \quad \mathbf{D_d} = \mathbf{D} \tag{5.5}$$

These matrices from (5.5) is not an approximation and will generate accurate results given that $u(t)$ is piecewise constant(doesn't change between the samples), If $u(t)$ is generated by a computer this usually isn't a problem. To avoid calculating infinite data series the matrix $\mathbf{B_d}$ is formulated

$$\mathbf{B_d} = \mathbf{A}^{-1}(e^{\mathbf{A}T_s} - \mathbf{I})\mathbf{B} = \mathbf{A}^{-1}(\mathbf{A_d} - \mathbf{I})\mathbf{B} \tag{5.6}$$

Given that $\mathbf{A_d}$ is non-singular. By using the MATLAB function
`[A_d,B_d,C_d,D_d] = c2d(A,B,C,D,T)`, the system in (5.3) will be transformed into the system in (5.4).

5.3.1 Internal stability of discrete LTI system

The MPD from section 2.5 with the parameters from (5.1) are discretized using the discretization from section 5.3. Which denotes the characteristic polynomial

$$\det(\mathbf{A_d} - \lambda\mathbf{I}) = 0$$

Where the roots are.

λ_1	λ_2	λ_3	λ_4	λ_5	λ_6	λ_7	λ_8	λ_9
1.0000	0.9745	0.9062	0.8169	0.7376	0.3181	0.2408	0.2872	0.2589

Which is satisfying theorem 1.2 and the system is marginally stable.

Theorem 1. [Chen]

1. *The equation* $\mathbf{x}[k+1] = \mathbf{A_d}\mathbf{x}[k]$ *is marginally stable if and only if all the eigenvalues of* $\mathbf{A_d}$ *have a magnitude less then or equal to 1 and those equal to 1 are simple roots of the minimal polynom of* $\mathbf{A_d}$

2. *The equation* $\mathbf{x}[k+1] = \mathbf{A_d}\mathbf{x}[k]$ *is asymptotically stable if and only if all the eigenvalues of* $\mathbf{A_d}$ *have a magnitude less then or equal to 1*

5.4 Constrained reference tracking MPC design

Consider the discrete-time linear time-invariant input affine system of a MPD represented in section 2.5 and discretizated using the technique represented in section 5.3. Where the controlled output is the bottom hole pressure p_1.

$$\begin{aligned} \mathbf{x}_{k+i+1|k} &= \mathbf{A_d}\mathbf{x}_{k+i|k} + \mathbf{B_d}\mathbf{u}_{k+i|k} + \mathbf{E_d}\mathbf{v}_{d_{k+i|k}} + \mathbf{B_{d,bias}} \\ \mathbf{y}_{k+i|k} &= \mathbf{C_{d,MPC}}\mathbf{x}_{k+i|k} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \mathbf{x}_{k+i|k} \end{aligned} \tag{5.7}$$

And the disturbance v_d, represented in section 3.1 is assumed known (predicted/measured). While fulfilling the constraints

$$y_{min} \leq \mathbf{y}_{y_{k+i|k}} \leq y_{max} \quad u_{min} \leq \mathbf{y}_{u_{k+i|k}} \leq u_{max} \tag{5.8}$$

At all time instants $k > 0$. Where the constraint on $y = p_1$ is denoted from the pore and fracture pressure from section 2.1, and the constraint on the input is denoted by the restrictions of the choke valve. The constrained reference tracking MPC solves the following quadratic optimization problem

$$\min_{u,x} \quad J(u,x) = \sum_{i=1}^{N} \left\{ \begin{array}{c} \mathbf{u}_{k+i|k}^{\top} R \mathbf{u}_{k+i|k} \\ +(\mathbf{y}_{k+i|k} - \mathbf{r}_{k+i|k})^{\top} Q (\mathbf{y}_{k+i|k} - \mathbf{r}_{k+i|k}) \end{array} \right\}$$

$$\begin{aligned} \text{Subject to} \quad & \mathbf{x}_{k+i+1|k} = \mathbf{A_d}\mathbf{x}_{k+i|k} + \mathbf{B_d}\mathbf{u}_{k+i|k} + \mathbf{E_d}\mathbf{v}_{k+i|k} + \mathbf{B}_{d,\text{bias}} \\ & \mathbf{y}_{k+i|k} = \mathbf{C_d}\mathbf{x}_{k+i|k} \\ & \mathbf{x}_k = \mathbf{x}[k] \\ & \mathbf{y}_{k+N|k} = \mathbf{r}_{k+N|k} \\ & y_{min} \leq \mathbf{y}_{y_{k+i|k}} \leq y_{max} \\ & u_{min} \leq \mathbf{y}_{u_{k|k}} \leq u_{max} \end{aligned} \tag{5.9}$$

at each sampling instant k. Where N is the length of the finite horizon in samples, J is the cost function, r is the reference trajectory, $\mathbf{x}[k]$ is the initial condition at time instant k, $r_{k+N|k}$ is the invariant terminal constraint, Q and R are positive definite.

5.4.1 Condensed formulation

In this section the problem is reformulated to a form which is implementable on a QP solver. The objective is to create an optimization problem where the only decision variable is $u_{k+i|k}$, this is often referred to as a condensed formulation as an opposition to a sparse/none-condensed formulation where both the states and the inputs are considered as decision variables in the optimization. This condenses the optimization problem in (5.9) into

$$\begin{aligned} \min \quad & u^{\top} H u + (F\theta + f)u \\ \text{s.t.} \quad & A_{ieq} u \leq b_{ieq} + B_{ieq}\theta \end{aligned}$$

where there are many undefined matrices which has to be chosen in such a way that they fit the problem from (5.9), but can be manipulated to some extent to fit the application of the controller. The main thing one can change is the initial condition x_0, and by switching this variable the type of controller changes drastically. If this variable for an example is chosen to be $x_0 = \mathbf{x}[k]$ it becomes impossible to change the reference. In the sections that follows a couple of different choices of initial conditions are presented.

Reference Tracking MPC

The first possibility to choose an initial condition θ is to include both the current state estimate and the desired reference trajectory r. Which will produce a reference tracking MPC.

$$
\begin{aligned}
F &= \begin{bmatrix} 2(P_1^\top P_5^\top Q P_5 P_2)^\top & -2(Q P_5 P_2)^\top \end{bmatrix} \\
\theta &= \begin{bmatrix} x_{k|k} \\ r \end{bmatrix} \\
f &= 2(P_3^\top P_5^\top Q P_5 P_2)^\top v_d + 2P_4^\top P_5^\top Q P_5 P_2 \\
A_{ieq} &= \begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix} \\
b_{ieq} &= \begin{bmatrix} d_u \\ d_y - D_y P_5 P_3 v_d - D_y P_5 P_4 \end{bmatrix} \\
B_{ieq} &= \begin{bmatrix} 0 & 0 \\ -D_y P_5 P_1 & 0 \end{bmatrix}
\end{aligned}
$$

Reference Tracking MPC with disturbance feed forward

Another option is to choose the initial condition θ to include the system disturbance. This will create a disturbance feedforward and the controller will be more capable to contract the the disturbances generated from the waves. The problem will of course be that this require some

sort of knowledge about the disturbance.

$$
\begin{aligned}
F &= \begin{bmatrix} 2(P_1^\top P_5^\top Q P_5 P_2)^\top & -2(Q P_5 P_2)^\top & 2(P_3^\top P_5^\top Q P_5 P_2)^\top \end{bmatrix} \\
\theta &= \begin{bmatrix} x_0 \\ y_{ref} \\ v_d \end{bmatrix} \\
f &= 2P_4^\top P_5^\top Q P_5 P_2 \\
A_{ieq} &= \begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix} \\
b_{ieq} &= \begin{bmatrix} d_u \\ d_y - D_y P_5 P_4 \end{bmatrix} \\
B_{ieq} &= \begin{bmatrix} 0 & 0 & 0 \\ -D_y P_5 P_1 & 0 & -D_y P_5 P_3 \end{bmatrix}
\end{aligned}
$$

System matrices in condensed MPC

When using a condensed QP formulation to solve a dynamical optimization problem, the state-space model from (5.7) have to be written out recursively and added to the cost function, because this process is a bit hairy it's moved to appendix A, but the resulting matrices becomes.

$$
\begin{aligned}
x &= P_1 x_0 + P_2 u + P_3 v_d + P_4 \\
y &= P_5 x \\
&= P_5 P_1 x_0 + P_5 P_2 u + P_5 P_3 v_d + P_5 P_4
\end{aligned}
$$

Where

$$P_2 = \begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \cdots & B \end{bmatrix}$$

$$P_3 = \begin{bmatrix} E & 0 & 0 & \cdots & 0 \\ AE & E & 0 & \cdots & 0 \\ A^2E & AE & E & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}E & A^{N-2}E & A^{N-3}E & \cdots & E \end{bmatrix}$$

$$P_4 = \begin{bmatrix} B_{bias} & 0 & 0 & \cdots & 0 \\ AB_{bias} & B_{bias} & 0 & \cdots & 0 \\ A^2B_{bias} & AB_{bias} & B_{bias} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}B_{bias} & A^{N-2}B_{bias} & A^{N-3}B_{bias} & \cdots & B_{bias} \end{bmatrix}$$

$$P_5 = \begin{bmatrix} C & 0 & \cdots & 0 \\ 0 & C & \ddots & \vdots \\ \vdots & \ddots & C & 0 \\ 0 & \cdots & 0 & C \end{bmatrix}, \quad P_1 = \begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}$$

And the MPC constraints from (5.8) is formulated into the constraint matrices.

$$
\begin{aligned}
d_y &= \begin{bmatrix} \begin{bmatrix} y_{max,1} \\ -y_{min,1} \end{bmatrix}^\top & \begin{bmatrix} y_{max,2} \\ -y_{min,2} \end{bmatrix}^\top & \cdots & \begin{bmatrix} y_{max,N-1} \\ -y_{min,N-1} \end{bmatrix}^\top \end{bmatrix}^\top \\
d_u &= \begin{bmatrix} \begin{bmatrix} u_{max,1} \\ -u_{min,1} \end{bmatrix}^\top & \begin{bmatrix} u_{max,2} \\ -u_{min,2} \end{bmatrix}^\top & \cdots & \begin{bmatrix} u_{max,N-1} \\ -u_{min,N-1} \end{bmatrix}^\top \end{bmatrix}^\top \\
D_y &= \text{diag}\begin{pmatrix} D_{y_1} & D_{y_2} & \cdots D_{y_{N-1}} \end{pmatrix} \\
D_u &= \text{diag}\begin{pmatrix} D_{u_1} & D_{u_2} & \cdots D_{u_{N-1}} \end{pmatrix} \\
D_{u_k} &= D_{y_k} = \begin{bmatrix} 1 & -1 \end{bmatrix}^\top
\end{aligned}
\tag{5.10}
$$

5.4.2 Slack variable

Feasibility of the optimization problem in (5.9) at each time instant k must be ensured, as discussed in section 4.4.2. To ensure feasibility at each time instant a slack variable can be added on the hard constraint of the drill bit pressure $y_{k+i|k} = p_1$ to create a soft constraint on the controlled variable.

$$
y_{min} - \varepsilon \leq \mathbf{y}_{y_{k+i|k}} \leq y_{max} + \varepsilon \tag{5.11}
$$

The slack variable ε is then added as a third decision variable in the cost function in (5.9), which generates a new cost function for the system with slack variables.

$$
\min_{u,x,\varepsilon} \quad J(u,x) = \sum_{i=1}^{N} \left\{ \begin{array}{c} u_{k+i|k}^\top R u_{k+i|k} \\ +(y_{k+i|k} - r_{k+i|k})^\top Q (y_{k+i|k} - r_{k+i|k}) \\ +\varepsilon^\top S \varepsilon \end{array} \right\} \tag{5.12}
$$

This also creates a new tunable parameter S, which determines the hardness of the constraint. It is ordinary to place a unnecessarily high weight on this variable because breaking shall not occur during normal operation, and if it breaches, it should be to ensure feasibility.

5.4.3 Controller tuning

Tuning an MPC controller is more a work of art than a craft, and it is a matter of experience to get it right. The flowchart in figure 5.3 suggest a possible procedure, where one starts with setting a

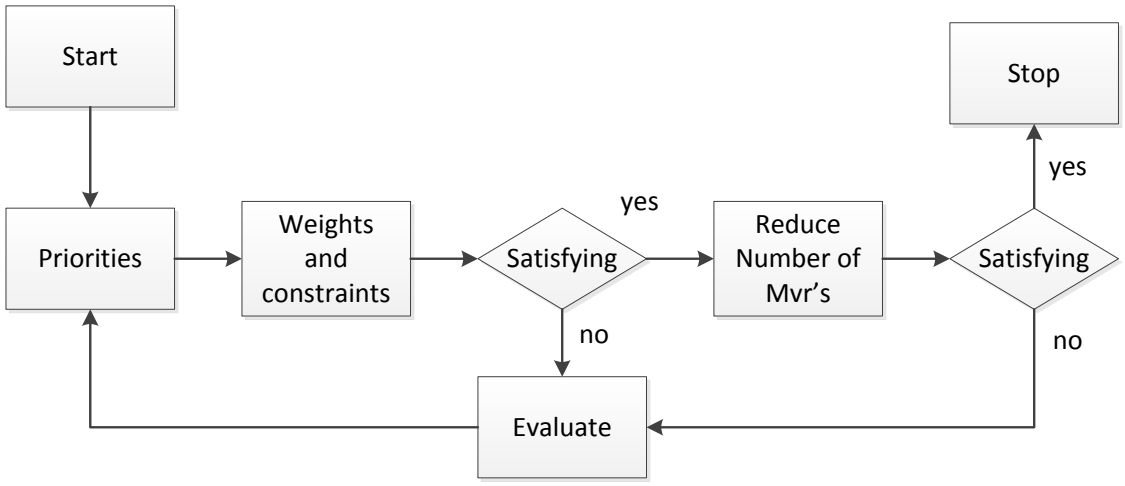


Figure 5.3: Simple tuning procedure

high sampling rate in the MPC and guess some weights suitable for the controller. Then

1. Simulate the process with MPC controller.
2. Check if the controlled variable converges to the reference and if the manipulated variable is used unnecessarily much.
3. Upgrade the weights if the result doesn't live up to the expectations or go to the next step if results are good.
4. If the weights produces satisfying results step up the sampling rate until the results starts to degrade.

Chapter 6

Implementation

The first sections in this chapter will be building up towards the MPC implementation by describing some of the concepts and tools used in the implementation. And in the end of the chapter the implementation itself is going to be explained in more detail.

6.1 Compiling code

Computers are advanced machines and in order to make them simpler they are built to take simple commands at a high speed. This is done by creating a simple language called *machine code*, which is so simple it's hard to write and easy to make errors. In fact, in the first computers the cost of making software often cost tree times as much as the computer itself. This gave birth to what's today called high-level *programming languages*. This language is usually quite different from the machine code and often inspired by a mixture of logic and English syntax. To make the computer understand the high-level language a compiler is used for translating the code into machine code. Some of the main reasons for using a high-level computer language might be.

- The structure, syntax and logic is closer to how humans think.
- The high-level code tends to be shorter than the equivalent machine code.
- The compiler can spot the most dubious mistakes.

- The same code can be compiled to many different machines.

This of course makes high-level computer code faster to write and cheaper. On the other hand, code written in high level languages and compiled to machine code tends to be a bit slower than hand coded machine code. Therefore machine code is still used today for some critical problems. For more information about compilers and basic compiler design one might reed (Ægidius Mogensen, 2010).

6.1.1 Makefiles

Programming can be put into a fairly simple routine. First edit your source files, compile them then debug the result. Although this sounds simple enough the programmer might use enormous amounts of time tracking down an error that didn't really exist. Or trying to fix a bug, but when you fixed the bug you where still running the old file. The problem of building executable files also have a tendency to grow more complex as the program grows and with the introduction of libraries.

A Unix program called *make* was intended to automate the transformation from source code to executable. "*make* make defines a language for describing the relationships between source code, intermediate files, and executables. It also provides features to manage alternate configurations, implement reusable libraries of specifications, and parameterize processes with user-defined macros. In short, make can be considered the center of the development process by providing a roadmap of an application's components and how they fit together."(Mecklenburg, 2004)

The make program runs a text file usually called makefile containing the specifications of how to run the program, which compiler to use, what library's to include and so forth. The source files is compiled into binary files, then merged together by a linker to form an executable program. (Mecklenburg, 2004)

6.1.2 CMake

CMake is an open-source, cross platform build system. This system is used to control the software compilation. This is done by adding a second layer of makefiles. You write a makefile for the CMake environment, then you run that makefile in CMake. This creates a new makefile specific for your environment. The benefits with this approach is that one makefile can be written to work on multiple operating systems with multiple compiler choices.

This program is used for building source code that includes the *acado* library. A simple guide, how to begin can be found at

http://sourceforge.net/p/acado/wiki/Using%20CMake%20-%20UNIX%20-%20Common/

For further information about CMake go to

http://www.cmake.org/

6.2 Matlab engine

The matlab engine is a library for C,C++ or Fortran. It is created in such a way that you can call Matlab software from your own programs. In this way you can use Matlab as a computational tool or simply as a plotting tool. To use this tool you must have Matlab installed on your computer, a version of Matlab Compiler runtime doesn't cut it.

Standalone programs written in C, C++ or Fortran communicate with a separate MATLAB process via pipes on unix systems, or via COM interfaces on windows systems. This library allows you to use all the common Matlab functions.

You can split the usage of Matlab engine into three groups

1. **Opening Matlab engine**

   Before you can use the Matlab engine you have to open one or more engine applications.

This is done by creating a pipe to Matlab as shown below

Engine *ep = engOpen(NULL)

2. **Runing matlab comands**
   Matlab scripts or .m files is based on running command lines in a script or separately in chronological order to do the desired job. In Matlab engine you can do the same, you basically send a command to the Matlab command window like this

   engEvalString(ep, "The matlab command")

   This way you can run either simple commands, Matlab script, Matlab functions or simulink models.

3. **Getting/Setting variables**
   To use the engine it might be useful to set variables to the Matlab engine

   engPutVariable(ep, "T", T);

   Or get variables d = engGetVariable(ep, "d");

   These variables are in a special matrix format and one need to go via them to get a ordinary data type.

All the information abut the Matlab engine can be found in the Matlab External Interfaces manual (MAT, 2014).

6.3 Acado toolbox

The simplest way of describing what acado toolkit is, is probably by using their own words from (David Ariens, 2014).

> ACADO Toolkit is a software environment and algorithm collection written in C++ for automatic control and dynamic optimization. It provides a general framework for using a great variety of algorithms for direct optimal control, including model predictive control as well as state and parameter estimation. It also provides (stand-alone) effciently implemented Runge-Kutta and BDF integrators for the simulation of ODE's and DAE's.
>
> —(acado toolbox)

The ACADO toolbox provides a MATLAB interface which makes the ACADO algorithms accessible from Matlab. An alternative to this is to implement the optimization problem as self contained C++ code. In addition to offer MPC algorithms for simulation purposes, it provides a code generation package for fast MPCs. This will generate C-code black box providing the optimization algorithm. This black box uses an iterative active set method with gauss-newton Hessian approximation. To iteratively solve the SQP/QP problem with a predefined number of iterations. All these iterations do not have to be iterated if the KKT tolerance criterion is fulfilled. A standard setup for this tolerance criterion is to choose a maximum tolerance of $1e^{-6}$

6.3.1 Code generation

The idea behind ACADO code generation is to write a program describing a non-linear model predictive control problem in C++, which is creating a new C function which solves the optimization problem. The dynamical optimization problem to be solved is presented on the form

below.

$$\min_{\substack{x_0,\dots,x_N \\ u_0,\dots,u_{N-1}}} \quad \sum_{k=0}^{N-1} \left\| h(x_k,u_k) - \tilde{y}_k \right\|^2_{W_k} \left\| x(x_N) - \tilde{y}_N \right\|^2_{W_N}$$

$$\begin{aligned}
\text{Subject to} \quad & x_0 = \hat{x}_0 \\
& x_{k+1} = F(x_k, u_k, z_k), \text{ for } k = 0,\dots,N-1 \\
& x_k^{\text{lo}} \leqslant x_k \leqslant x_k^{\text{hi}}, \text{ for } k = 0,\dots,N-1 \\
& u_k^{\text{lo}} \leqslant u_k \leqslant u_k^{\text{hi}}, \text{ for } k = 0,\dots,N \\
& r_k^{\text{lo}} \leqslant r_k(x_k,u_k) \leqslant r_k^{\text{hi}}, \text{ for } k = 0,\dots,N-1 \\
& r_N^{\text{lo}} \leqslant r_k(x_n) \leqslant r_N^{\text{hi}}
\end{aligned}$$

Where

- $x \in \mathbb{R}^{n_x}$ Is the differential states
- $u \in \mathbb{R}^{n_u}$ Is the control input
- $\hat{x}_0 \in \mathbb{R}^{n_x}$ The current state measurement/estimate
- $h \in \mathbb{R}^{n_y}$ and $h_N \in \mathbb{R}^{n_{y,N}}$ Is the reference variables
- $W_k \in \mathbb{R}^{n_y \times n_y}$ and $W_N \in \mathbb{R}^{n_{y,N} \times n_{y,N}}$ Are the weighting matrices.
- $\hat{y}_k \in \mathbb{R}^{n_y}$ and $\hat{y}_N \in \mathbb{R}^{n_{y,N}}$ Is the reference vectors
- $x_k^{\text{lo}} < x_k^{\text{hi}} \in \mathbb{R}^{n_x}$ and $u_k^{\text{lo}} < u_k^{\text{hi}} \in \mathbb{R}^{n_u}$ are bounds on control inputs and states control bounds.

The newly created C code for solving a non-linear MPC problem uses a range of efficient algorithms in the optimization process. This code generation algorithm is set up for solving nonelinear MPC problems but by studying the generated code, it is found that if the algorithm is presented a linear problem it will generate a regular QP instead of an SQP algorithm.

6.4 PLCs

PLCs have their origin from logical relay control systems. These control systems where expensive to build or modify. So when the computers became easily available, it was naturally to implement these roles in a programmable unit. With time, more and more systems implemented in the PLC, and after a while the PID controllers where implemented as well. At today's oil platforms the process controlled by only a few PLCs with a plethora of RIO(remote i/o) connections.

Connectivity

These controllers are made to control processes so there has to be an interface to the industrial machinery. These connections are usually divided into inputs and outputs. Where both the inputs and the outputs are divided into digital and analogue connections. The digital i/o can for an example be used to turn on a pump or get a running signal. The analogue i/o is used for getting or setting process variables. Examples on analogue i/o can be setting the choke valve position or getting the topside oil pressure.

The PLCs are also usually connected to a human machine interface (HMI) for observation or interaction with the process.

Programming Languages

The most commonly used text-based programming languages for PLCs are *structured text* but in this thesis the programming language C is going to be used.

6.4.1 Program execution

There are different ways of executing a PLC program. The PLC programming languages are procedural, meaning the program goes through different stages throughout the program and execute the commands as they come. Cyclic execution is an example of a procedural program where all the blocks in figure 6.1 are stages the procedural program goes through.

1. **Initialize**

   Sets all the initial conditions of the program.

2. **Reads all inputs**

   Reads in the measurements from physical or network based i/o and puts it in the internal memory.

3. **Program**

   Runs PLC program. This might be all sorts of control problems, from logical control to advanced process control.

4. **Update all outputs**

   Gets the new data from memory and updates i/o

5. **Go to step 2.**

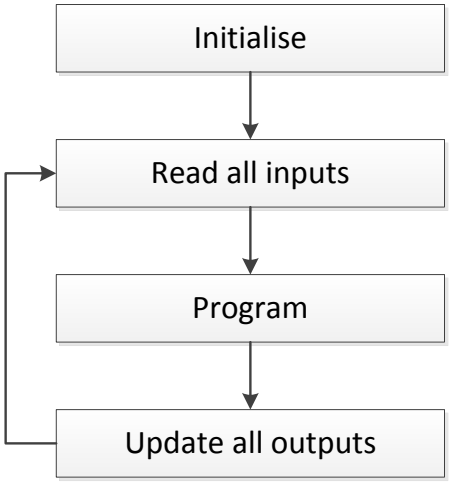


Figure 6.1: Cyclic execution program

6.5 Acado Implementation

6.5.1 Software Installation

ACADO is a native Linux application, and to run it in windows a Compatibility layer is needed. An alternative is to use Cygwin which gives Linux like environment. From a strictly practical per-

spective Cygwin installs a unix like terminal on the native windows environment. The Cygwin application can be downloaded for Free from

www.cygwin.com

Another approach is to use native Linux system and the author is using Linux Fedora OS, but the procedure is the same for Cygwin with the exception that the package management system "YUM" is switched to "apt-get". When a proper operating system or emulator is installed some software needs to be installed in order to install the ACADO toolbox.

Software	Description
gcc	C compiler.
g++	C++ compiler.
CMake	Software for managing build process.
Git	Revision control software.
gnuplot	Plotting software.
Doxygen	documentation generator a tool.
Graphviz	Graph Visualization Software

Before installing these programs note that you have to be logged on as a super user(root), which can be done by typing *su* in the terminal followed by the root password. To install the software listed above using the yum package manger, type.

```
yum install gcc g++ cmake git gnuplot doxygen graphviz
```

Now that all needed software is installed, the ACADO toolbox can be downloaded from github using the pre-installed software GIT. Before running the command below it's important to maneuver into a desirable location.

```
git clone https://github.com/acado/acado.git -b stable ACADOtoolkit
```

Further more enter the newly created "ACADOtoolkit" folder and build the newly downloaded software.

```
cd ACADOtoolkit
mkdir build
cd build
```

```
cmake ..
make
```

To test if ACADO is properly installed run the test problem.

```
cd ..
cd examples/getting_started
./simple_ocp
```

Now that most of the programming environment is set-up the only missing thing is a (C/C++) editor of your choice. An alternative is to use the eclipse editor with with CDT(C/C++ Development Tooling) extension. This program is free and easily installed in the terminal using the command.

```
yum install eclipse eclipse-cdt
```

Setting up the environment

We need to make the operating system aware of the ACADO toolbox in order to use the library outside the ACADO folder. One way of doing this is by adding this is to edit the ".bashrc" file. This is a script file that contains commands that will be executed at start-up in a Linux environment, and is often used to add directories to PATH or setting up environment variables. Because this file is hidden the easiest way of opening it is through the terminal, so go to the home folder on your system and run the command

```
gedit ~/.bashch
```

This of course implies that the text editor "gedit" is installed, but any text editor will do the job. Furthermore add the following line in the bottom of this file then save and close it.

```
source "Acado Root Folder"/build/acado_env.sh
```

To implement the changes you now have made to the system environment run the command or restart the system.

```
. ~/.bashrc
```

6.5.2 Generate MPC Solver

Setting up a project

We begin with creating a new empty project with an empty source file. To show how this is done, an image tutorial on how to create a new project in eclipse is added under appendix C.1.

Now that we have created our project we need a makefile to compile an executable. The makefile is created by a cross platform build managing program called CMake. The idea behind this is to create a makefile that contains the information needed to create makefiles in different environments. To link the project folder to the ACADO toolbox, copy the file "FindACADO.cmake" from "/cmake/FindACADO.cmake" in the ACADO folder to the project folder. Now create a new empty ".txt" text file in the project folder. This is the settings file where all the build settings are defined. The settings file used in this project is added under scripts in appendix D.1.1. Now create a new folder inside the project folder called "build". Enter this folder in the terminal and type "cmake .." to create the makefile, and type "make" to compile the project and create an executable.

In order to efficiently be able to write source code, the build process needs to be implemented in the editor. This is also a bit hard to explain how to do in a few words so this is also explained in an image tutorial in appendix C.2.

Write code generation source code

This section is going to contain the main points in the QP code generation script based on the controller from section 5.4, but for a complete script take a loop at appendix D.1.2. For more information about the code generation or ACADO in general please refer to (David Ariens, 2014).

We start of by defining the differential states of the MPD system from chapter one, plus one differential state "dummy" which is just implemented in order to be able to add the slack variable to the cost function.

```
DifferentialState    p_1;
```

```
DifferentialState   q_1;
DifferentialState   p_2;
DifferentialState   q_2;
DifferentialState   p_3;
DifferentialState   q_3;
DifferentialState   p_4;
DifferentialState   q_4;
DifferentialState   p_5;

// Slack variable dummy state.
DifferentialState dummy;
```

Defines the control input $u = u_a = q_{bpp} - q_c$ and a slack variable defined as a control input.

```
Control  u; // Control input
Control  s; // Slack variable
```

The system parameters can be implemented as parameters, and this way one can change the parameters online. But because the depth is changing so slowly, creating simulations where the depth is changing would be meaningless, so the parameters will be defined as constants. Where for these simulations $a_i = a$, $b_i = b$, $c_i = c$ and $e_i = e$ for all values of i.

```
const double a = 2.25;
const double b = 4.28;
const double c = 14.5;
const double e = 2.64;
const double Kc = 8.96;
const double p0 = 1.01;
const double qbpp = 14.88;
```

And to implement the measured disturbance into the differential equations a parameter vd is added to the system, this variable can be set from the simulator or MPC program.

```
parameter vd;
```

Now that all the inputs, differential states, parameters and disturbances are defined we can denote the differential equations from section 2.3, into the MPC from section 5.4.

```
DifferentialEquation f;
f << dot( p_1 ) == a*(-q_1 - vd*0.0656*21);
```

```
  f << dot( q_1 ) == b*(p_1-p_2) - c*q_1-e;
  f << dot( p_2 ) == a*(q_1-q_2);
  f << dot( q_2 ) == b*(p_2-p_3) - c*q_2-e;
  f << dot( p_3 ) == a*(q_2-q_3);
  f << dot( q_3 ) == b*(p_3-p_4) - c*q_3-e;
  f << dot( p_4 ) == a*(q_3-q_4);
  f << dot( q_4 ) == b*(p_4-p_5) - c*q_4-e;
  f << dot( p_4 ) == a*(q_4 + u);
// Slack variable defined in dummy state
// so it can be implemented in cost function.
  f << dot(dummy) == s;
```

The reference function h from the optimization problem in section 6.3.1 from the MPC formulation in section 5.4, is defined as $h = \begin{bmatrix} u & p_1 & s \end{bmatrix}$, and the terminal constraint is defined as $h_N = p_1$.

```
  Function h, hN;
  h << u << p_1; << s
  hN << p_1;
```

Where the corresponding weighting matrices is denoted as

```
Matrix W  =  eye( h.getDim()  );
  Matrix WN =  eye( hN.getDim() );
  W(0,0) = 15;       // Input weighting
  W(1,1) = 150;      // Output weighting
  W(2,2) = 5000;     // Weight of slack
  WN(0,0) = 5000;    // Terminal weight
```

Where the slack variable is weighted hard to work as a soft constraint and the terminal constraint is weighted hard to satisfy $x_{k+N} = \omega = x[k+N]$ the invariant terminal constraint. The optimal control problem denoted above is then assembled in

```
  OCP ocp(0.0, N*Ts, N);
  ocp.subjectTo( f );
  ocp.minimizeLSQ(W, h);
  ocp.minimizeLSQEndTerm(WN, hN);
```

Where we define the constraints on the drill-bit pressure as

```
const double ymin = 280;
const double ymax = 250;
ocp.subjectTo(p_1 + s - ymax <= 0);
ocp.subjectTo( 0 <= p_1 + s - ymin);
```

The maximum constraint on the input is defined as $G(u) = 1$ and the minimum is defined as $G(u) = 0$ which gives the constraints.

```
ocp.subjectTo(0 <= u - qbpp + Kc*sqrt(p_5-p0) );
ocp.subjectTo( u -qbpp <= 0 );
```

Now that the MPC is completely defined we can generate the QP solver by.

```
OCPexport mpc( ocp );
mpc.set( INTEGRATOR_TYPE,              INT_RK4          );
mpc.set( SPARSE_QP_SOLUTION,        SPARSE_SOLVER       );
mpc.set( QP_SOLVER,                    QP_FORCES        );
```

Where there is a lot of options one can change, for a complete overview of the options read (David Ariens, 2014). For an example can the sparse solver be switched for a condensed solver, but then the solver needs to be switched to the "QP_OASIS" solver because "QP_FORCES" doesn't support sparse solutions.

6.5.3 Implement solver in Simulations

Setup simulator project

To create a system Simulator written in C, we begin by creating a new empty project using the same tutorial as when generating the MPC solver. This torturial can be found in appendix C.1. We will in this section show how to use the "QP_OASIS" solver with full condensing and therefore we need to copy the oasis solver source code into the project folder. To be more specific copy the folder called "qpoasis" from "acado root folder/external_packages" to the root folder in your project. Then copy the generated source files into the project. Note that to have something to start from it might be useful to add the generated makefile and source file to the code generation options.

```
mpc.set( GENERATE_TEST_FILE,          YES              );
mpc.set( GENERATE_MAKE_FILE,          YES              );
```

The generated source files should contain the files listed in table 6.5.3, if the test file and make-files are included in the options.

File name	File format
acado_auxiliary_functions	c
acado_auxiliary_functions	h
acado_common	h
acado_integrator	c
acado_integrator	c
acado_qpoases_interface	cpp
acado_qpoases_interface	hpp
acado_solver	c
Makefile	
test	c

Table 6.5.3: Files generated by the ACADO toolbox

To connect the simulator to the Matlab engine from section 6.2, the generated makefile needs to be modified to call the engine. To do this the following lines needs to be implemented into the makefile. For starters we add a path to the Matlab root folder.

```
MATLAB = /usr/local/MATLAB/R2013a/ // Path to matlab root folder
```

Then the Matlab engine library can be loaded by the command.

```
LDLIBS += -L/usr/local/MATLAB/R2013a/bin/glnxa64 -Xlinker -rpath -
    Xlinker (MATLAB)/R2013a/bin/glnxa64 -leng -lmx
CFLAGS += -I$(MATLAB)/extern/include -I$(MATLAB)/extern/include/cpp
CXXFLAGS += -I$(MATLAB)/extern/include -I$(MATLAB)/extern/include/cpp
```

And the environment variables can be passed to the compiler. Here is an example on how to add the environment variables both in C and C++, but in our program only the first line is needed.

```
CFLAGS += -I$(MATLAB)/extern/include -I$(MATLAB)/extern/include/cpp
CXXFLAGS += -I$(MATLAB)/extern/include -I$(MATLAB)/extern/include/cpp
```

To compile the generated code, we can open the terminal and manoeuvre to the project folder and type "make". Note that this is necessary to set up an executable in the editor. To setup the editor to use the makefile above to compile the code, use the tutorial in Appendix C.2. The complete makefile can be found in appendix D.2.1.

Write source code

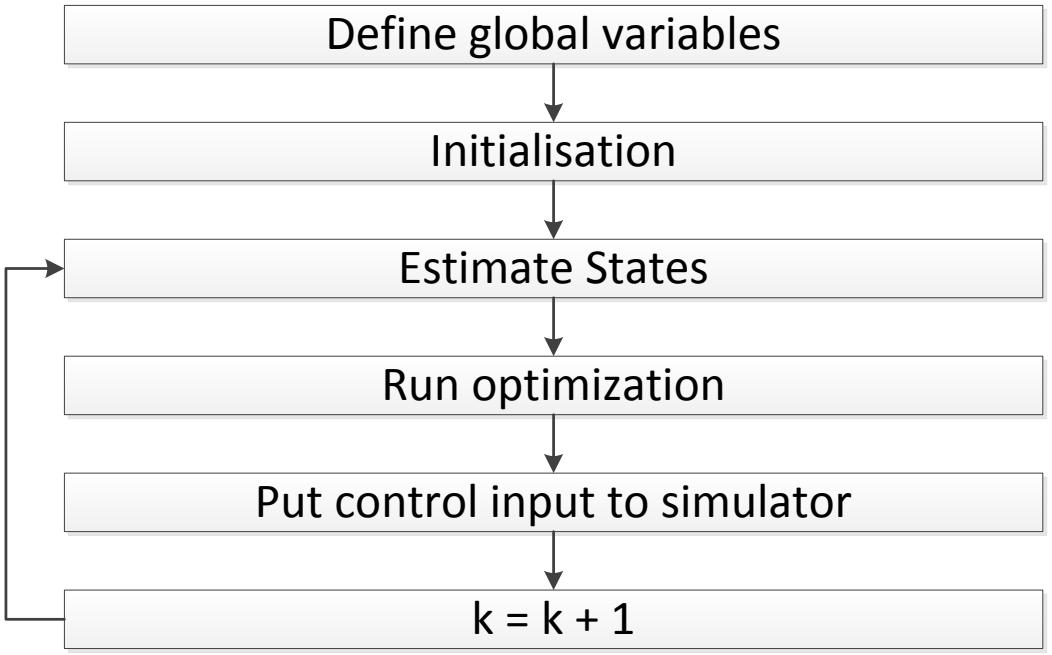


Figure 6.2: System simulator execution procedure

The system is chosen to be implemented as shown in figure 6.2, and the following section will explain the different steps.

We begin with **Defining the global variables**, which is done in the header file shown in appendix D.2.2. To reduce the number of variables in the project, most variables are defined as global, this is done because if variables are defined inside loops the PLC must constantly declare variables and this might effect the limited PLC memory and the system might crash. A summary of the variables needed for the simulation is enlisted below.

- To make the compiler aware of the functions they are declared in the header file.
- Get the optimization variables from the generated code.
- Define the simulation length and maximum number of iteration in the optimization.
- Define the state space matrices and all the plant information needed for the optimization or the state estimator.
- Define the matrices needed in the kalman filter.
- Define control, output, and disturbance variables.
- Define the Matlab engine variables used to get variable from the Matlab engine.

The **initialisation** is split into two functions, one for initializing the Matlab engine and one to initialize the optimization. The Matlab initialisation creates a bridge between a Matlab command window and the C program. It also initializes the plant initial values needed to run the plant simulator in Matlab. To get the detailed information abut the initial step, the functions are sown in appendix D.2.3.

The **optimization** is an implementation of the iterative QP solver from section D.2.5. Where we iterate the QP solver until a satisfying solution is found, and when the solution is found the optimal control input is returned.

The loop from figure 6.2 is the simulation iterator, where k denotes the sample number from the MPC in section 5.4. This loop is as one can see from the figure consisting of three different steps

1. Run the kalman filter appendix D.2.4, to estimate the states used in the MPC.
2. Run the MPC optimization for step k.
3. Put the optimal control input into the simulator in Matlab.
4. Go to the next iteration step $k = k + 1$

6.5.4 Implement solver in a PLC

From the previous section we have basically set up both the optimization QP and the simulator model, so this section will focus on how to implement this into an ABB AC500 PLC.

Rewriting the code to be PLC implementable

First, the ABB software doesn't support C++ code and the "QP_OASIS" solver is written in C++. To solve this problem the "QP_OASIS" solver is switched to the "QP_FORCES" solver, which doesn't support a condensed solution and therefore a sparse solver is chosen in the MPC settings.

```
mpc.set( SPARSE_QP_SOLUTION,        SPARSE_SOLVER       );
mpc.set( QP_SOLVER,                   QP_FORCES         );
```

This will give a Matlab file defining the solver problem, and to generate the solver we need to run this file in matlab. Note that this file isn't directly runnable without downloading the forces solver, which can be downloaded from

www.forces.ethz.ch/

Now that we have presumable run the Matlab script we get a forces folder containing a solver project, but we are only interested in the solvers source code files.

File name	File format
forces	.c
forces	.h

Then copy both files in table 6.5.3 and the forces files into a new folder to be implemented into a PLC function block.

At this point the program can't be implemented in the PLC software because of the following reasons. The PLC doesn't support print functions, illegal variable declaration, no makefile and the Matlab simulator. These problems needs to be addressed in order to implement the MPC.

- **Print functions**

  The generated files are written with a range of different print functions. In order for these files to be implementable, these needs to be commented out.

- **Illegal declarations**

  In the "forces.c" file some variables are declared as a sum of other variables, this notation is not accepted by the ABB compiler. To work around this problem one can make a new function with the sole purpose to make those summations, and remove the summation from the declaration itself. This function is then run in the solver initialisation. Note that there are hundreds of variables declared this way and it may take some time to implement these changes.

- **No makefile**

  The ABB build process isn't invoked by a user defined makefile explaining the build process. This means you can only use the libraries already established within the software, and you can't use linkers. To work around this problem we include all the source code files in the simulator header. Because we don't have an INCLUDE environment from the makefile, the includes needs to be added with question marks insisted of angel braces.

- **Matlab engine**

  Previously we have used the Matlab engine to do the simulations, but the PLC don't have access to Matlab. To solve this problem a simulator is written in C to do the process simulations inside the PLC. The C simulator is shown in appendix D.1.2.

Making a PLC MPC function block

First of we need to install the ABB Control builder software, when installing the PLC software just follow the installation manager and remember to cross of everything containing OPC servers, because its used in the PLC/PC connection.

Now that the ABB software package is installed the MPC, kalman filter and simulator can be implemented in a function block. To have common ground for implementation of the code into the function block, we presume that a project with an empty function block is created.

- Then the code files created in the section can be copied into function block folder. Note that the main code file needs to have the same name as the auto generated source file we

are replacing, and the main function declaration must be identical to the main function in the original function.

- If one wants any readouts from the simulations these needs to be defined as input, output or variables in the control builder software. This will automatically define the variables in the source code and can easily be written to.
- When the code is implemented in the ABB software and all the desired input and output variables are defined and used in the source code we can compile the project by pushing the compile button.

Running the PLC

By double clicking on AC500 in the device tree in the PLC program editor CoDeSys will automatically open. Because our program runs the hole simulation inside the function block there is no need of multiple iterations in the plc implementation. This is not a realistic implementation if it is going to be used on a physically plant, but it will give us data on the performance of the MPC controller in the form of running time, processing power and memory consumption. The function block can easily be implemented in the PLC structured text code by adding the block as a variable

```
PROGRAM PLC_PRG
VAR
        MPC: POU;  (*Defining the MPC function block*)
END_VAR
```

And in the main PLC loop the MPC function block is run by the code.

```
IF MPC.iterCounter < 1 THEN
        MPC();
END_IF;
```

Which will run the MPC one time as explained above, then do nothing for the rest of the iterations.

6.6 Matlab Implementation

To optimise the MPC controller and to have an MPC controller implementation independent from the ACADO MPC. A MPC controller implemented in Simulink is created. The block diagram for this simulation can be seen in figure 6.3. To explain how the control system is built up we are going to take a closer look at the different blocks.

- **Kalman filter**

  To estimate the states a kalman filter is used. The kalman filter is created from the discrete filter equations in section3.2 combined with a unit delay.

- Nonlinear state space model implementation of the equations in 2.3.

- The disturbance from section 3.1 is made into a disturbance vector and this vector is read into simulink from this vector.

- This block contains the feedback linearisation equations created in sec **??** In section 4.4, a condensed MPC was formulated. This MPC formulation is in this scenario implemented in a Matlab "S-function" and solved with the QP solver "quadprog".

```
[z,fval,exitflag,output,lambda] = quadprog(H,(F*uu+f)',Aieq,Bieq*theta
    +bieq,[],[],[],[],[],options);
```

  Where "theta" is defined as $\theta = \begin{bmatrix} u^\top & p_{1,\text{ref}}^\top & v_d^\top \end{bmatrix}^\top$, but can be manipulated to change the properties of the MPC as shown in section 5.4.

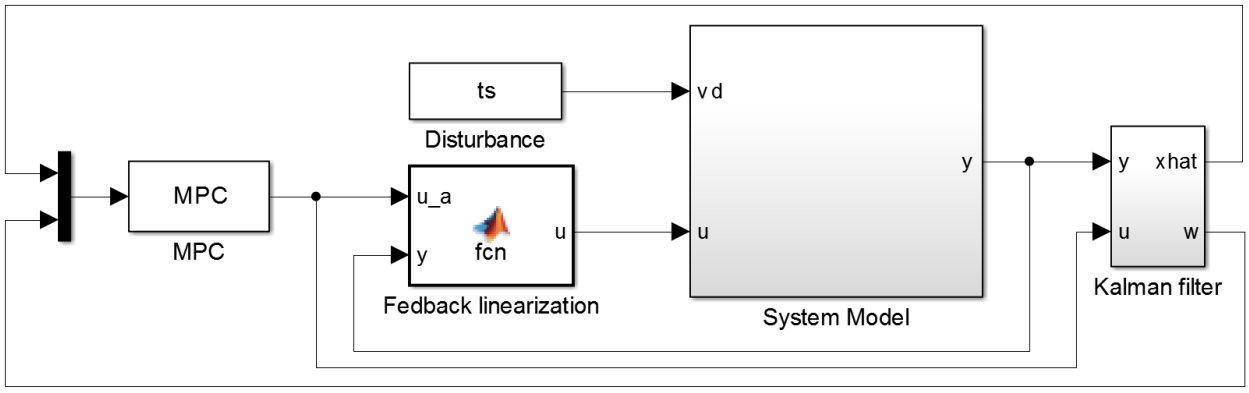


Figure 6.3: Simulink MPC implementation

Chapter 7

Simulation Results

In this section the set up for the simulations will be stated and the simulation results presented. The simulations will part by part test the different aspects of the problem, until the end where a PLC implementable MPC controller will be presented as a product of the results. For all the simulations the MPCs presented in section 5.4 will be used, and if not specified otherwise the parameter weights denoted below are used.

Parameter	value
Q	150
R	15

The parameter weights were found using the tuning strategy presented in section 5.4.3. Where it is assumed that the model parameters for a 1990.99 [m] long well presented in (5.1) is used. In section 5.2.3, it was shown that the MPD model with these parameters is controllable, observable and that the system is marginally stable.

7.1 Kalman filter

In most real processes measuring all the states is not a possibility, or even if it was possible would it be sensible to do so. The simple answer is probably no. If it was possible it would require an immense amount of instrumentation and would be extremely expensive, so in most cases the states needs to be estimated. There are many approaches to estimating the states, and the methodology should be chosen to fit the problem. In this case a Kalman filter is chosen

because there is prior knowledge about the disturbance and it's properties and the kalman filter has an excellent ability to filter disturbances. In section 3.2.2, a stability theorem for kalman filter stability was presented, and because the system is both controllable and stochastically reachable it shall in principal have filter convergence. To verify that the states convergence and analyse the performance of the Kalman filter a test scenario is set up in Matlab. For this scenario a ordinary condensed MPC is chosen, with the following specification.

Parameter	value
Q	150
R	15
T_{MPC}	1
N	15

The disturbance source

The disturbance is created from the drill-bit moving like a piston near the bottom of the well during pipe connections. This effect is created from the waves moving the rig in heave direction while the drill string is fixed to the drill floor. Consequently, the bottom hole pressure $p_{bit} = p_1$ is varying rapidly. This means the disturbance starts in p_1 and creates pressure waves beginning in the lowest control volume and spreading upwards. As can be seen in figure 7.1 sub-figure [2,1] and figure 7.2, the estimation error is also highest in the lower control volume and dissipating towards the surface. This makes the performance of the Kalman filter slightly more important because the disturbance directly affects the controlled variable in the MPC, and if this variable is wrongly estimated the MPC will control the system towards an invalid pressure.

Sample rate

The squared estimation error between the actual states and the estimated stated are plotted in figure 7.1 and 7.2 can be formulated as.

$$e_{i,k}^2 = (x_{i,k} - \hat{x}_{i,k})^2$$

Where i denotes the state in question and k the time instant. In this formula all the data is acquired from the same time instant, and from a quick look at the mathematical formulations

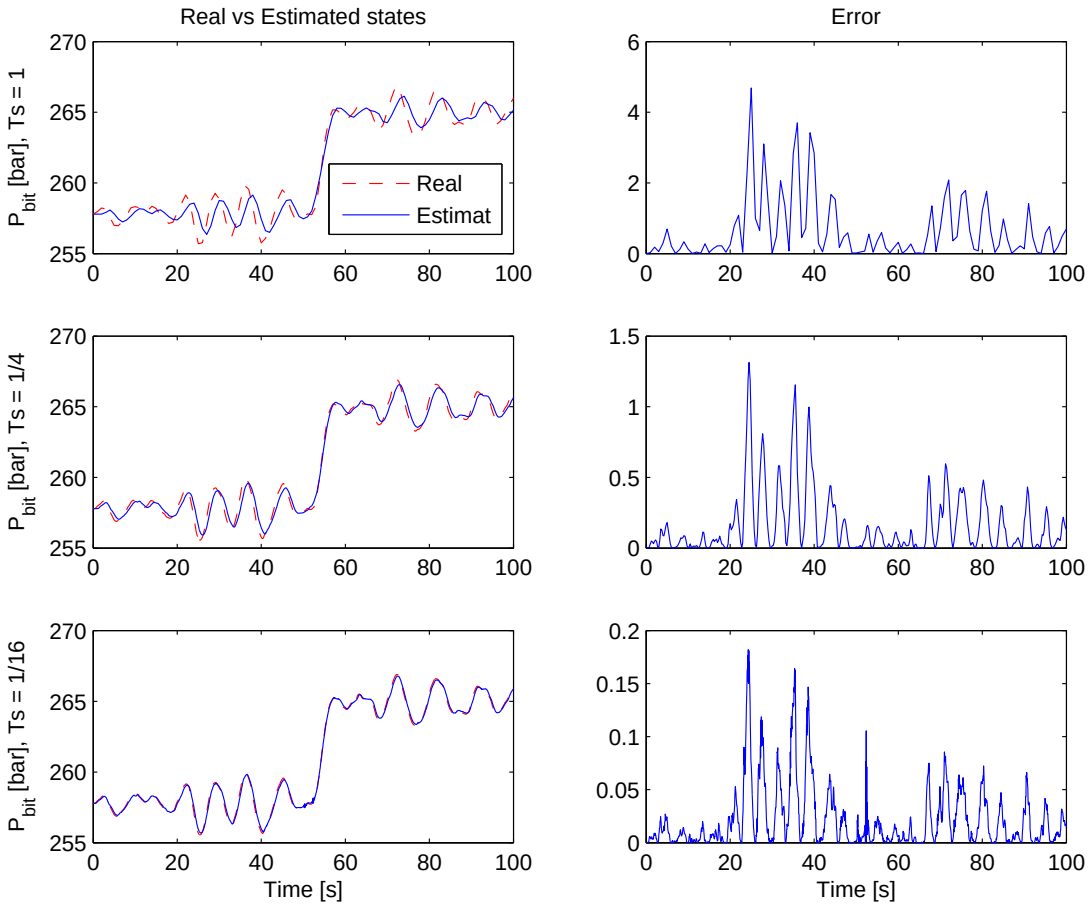


Figure 7.1: Estimation errors and sampling time

behind the kalman filter it is not an obvious correlation between updating rate and estimation error. And if the disturbance had been completely random it wouldn't matter. But if one think about it the disturbance isn't completely random, there is a correlation between e_k and e_{k+1}. So in principle if one increase the sample rate, there is more samples to estimate the same disturbance, or in other words more iterations to converge towards the solution. Consequently if the sampling rate is increased the estimation error will decrease, as can be seen in figure 7.1. For this reason it's always a good policy to select the sampling rate as high as the measurement sampling rate, a higher sampling rate than measurement sampling rate will be a waste of resources because it does not bring any new information to the table. In figure 7.1, one can clearly see this effect. When the sampling time is decreased from $T_s = 1$ to $T_s = 0.25$ the estimation is approximately four times lower, and likewise when the sampling time is decreased from $T_s = 0.15$

$T_s = 0.0625$ the estimation error is approximately four times lower. From these observations it seems like there is a linear dependence between estimation error and sampling time.

Initials conditions

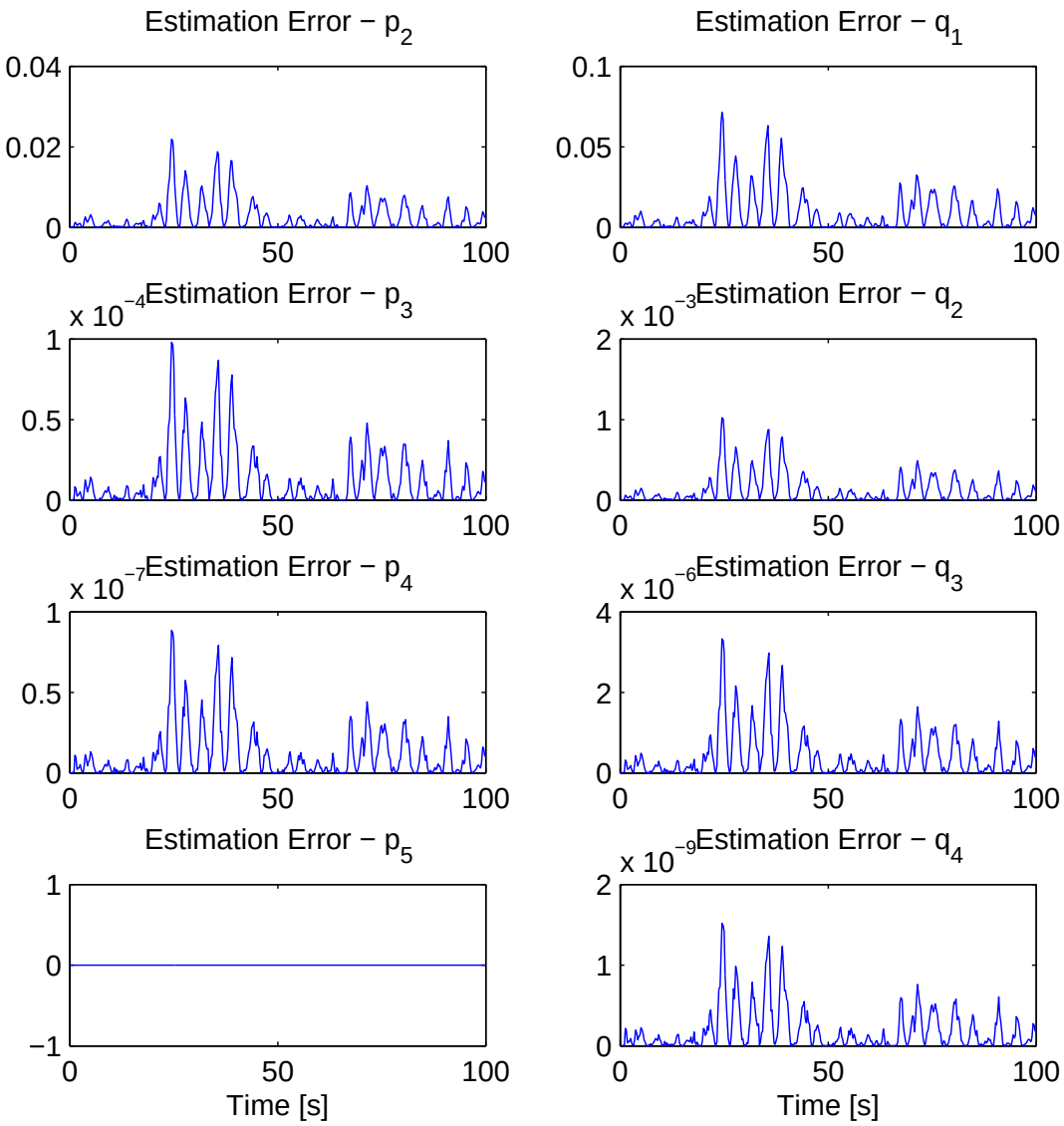


Figure 7.2: Estimation errors in different states white, $T_s = 1/4$

Especially when doing short simulations like in a thesis it's important to choose good initial conditions for the estimation. Even if the kalman filter has good convergence capability's it has

some form of convergence time until all the estimation errors are reasonably small. This may cause some problems for the controller in the beginning of the control problem, and one should be aware of it. In this thesis, the initial conditions for the simulator is found by setting presuming that all the states are in seedy state and that there is no flow between the the control volumes. This forms the pressure equation $p_i = p_{i+1} + e_i / b_i$ where the topside pressure is denoted by the choke equation. Which with the parameters from (5.1) gives the initial states are denoted as

$$x_0 \approx \begin{bmatrix} 257.8 & 0 & 196.1 & 0 & 134.4 & 0 & 72.7 & 11 \end{bmatrix}^\top$$

Because the initial conditions from the simulator is exactly known, the initial conditions in the Kalman filter is presumed known and this will not be a problem for this simulations. But we are still missing the information about the initial covariance matrix P_0^- and should have it in the back of our heads.

Disturbance estimation

Even if the kalman filter is commonly used for state estimation it can also be used for estimation of disturbances. Estimating measurement disturbances is very easily done and for this reason it is also a very common thing to do. The only thing one need to do is turn around the output formula from the state space formulation and we get.

$$\hat{v}_k = y_k - C_d \hat{x}_k$$

This is notably unnecessary for our process when we don't have any measurement disturbance, but we get an idea how to do the same thing for process disturbances. One way of implementing this is by subtracting the priori estimate from the states, even if this only gets the disturbance at the last sample.

$$\begin{aligned} \hat{x}_k^- &= A_d \hat{x}_{k-1} + B_d u_{k-1} \\ x_k &= A_d x_{k-1} + B_d u_{k-1} + w_{k-1} \\ x_k - \hat{x}_k^- &= w_{k-1} \end{aligned}$$

T_{MPC}	Horizon [sec]	Horizon [N]	Sparse variables	Condensed variables
0.1	15	150	1509	150
0.3	15	50	509	50
0.6	15	25	259	25
1.5	15	10	109	10
3	15	5	59	5
5	15	3	39	3

Table 7.1: Different choice of control horizons

Where the states are unknown, and the estimated states $\hat{x}$ are used in stead. Which forms the estimated disturbance $\hat{x}_k - \hat{x}_k^- = \hat{w}_{k-1}$. To do this the estimated states must be presumed $\hat{x_k} \approx x_k$ for this to be accurate. This way there is possible to create an estimate of the process disturbance w_{k-1}. But in reality we need w_k and in order for this to be accurate, the update rate needs to be chosen to be low $T_s \to 0$ because then $w_k \to w_{k+T_s}$. So if we chose $T_s = 1/16$ we get the disturbance estimation shown in figure 7.3. And we see that although the disturbance is from the last sample it gives a good estimate because the sample time is chosen to be of a negligible size.

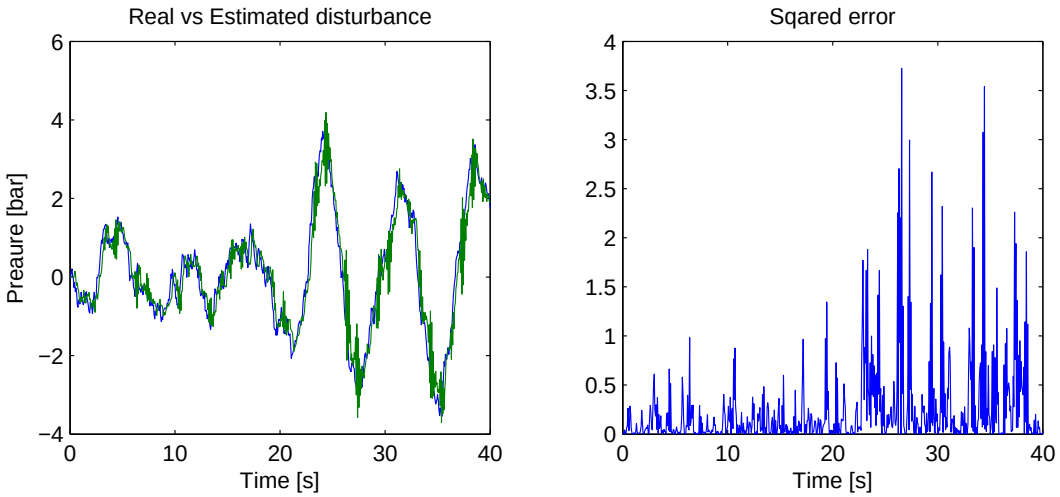


Figure 7.3: Estimated vs real disturbance

7.2 Heave Disturbance

In almost all the following scenarios the disturbance generated by the rigs motion in heave direction from section 3.1 is going to have an influence on the system behaviour. To easier depict how much this disturbance influence the system dynamics it is useful to plot how much pressure abnormalities this disturbance is adding to the button hole pressure. Figure 7.4 shows the pressure added to the bottom hole pressure by the disturbance. From this figure it is quite clear that the disturbance is oscillating between minimum -5 to maximum 5 [bar]. Which would essentially mean that if there is no disturbance rejection at all, the system will have an oscillation on the bottom hole pressure of ± 5 [bar]

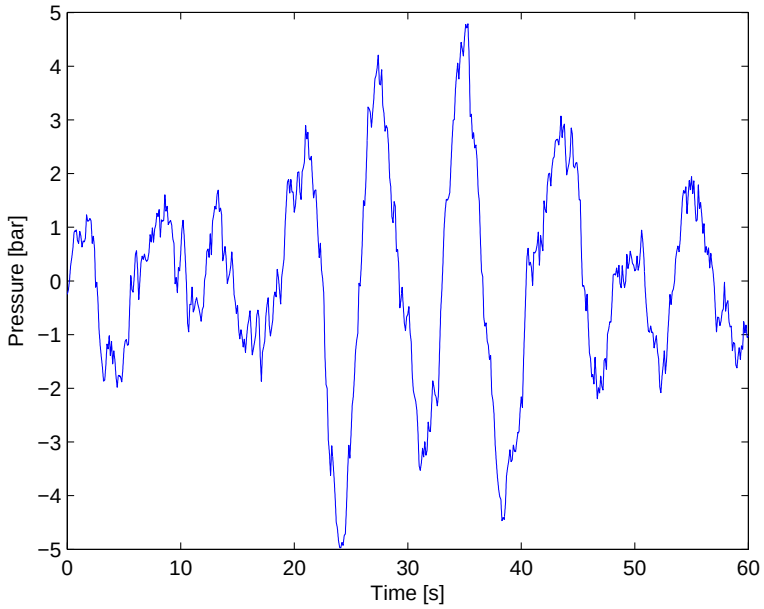


Figure 7.4: Disturbance influence pressure

7.3 Selection of control horizon and MPC sampling frequency

When designing a MPC controller one of the most important things to consider is the length of the control horizon. The control horizon should be chosen sufficiently long so as to stabilize the system. In most cases it is chosen so long that the system reach steady state. From section 4.4 we see that in order to guarantee stability we must guarantee that the last step in the control horizon must be within the control horizon to satisfy the invariant terminal constraint. This is commonly implemented by either putting an equality constraint on the end of the control horizon as done in section 5.4, or weighting the optimization extra high as done in the ACADO implementation section 6.5. Either way it is important to choose a control horizon that is sufficiently long. If the control horizon is chosen to short, the control effort must become larger to satisfy the terminal constraint, and in worst case it might turn into a feasibility problem. For these experiments it is important to find a system which stabilises the down hole pressure at all reasonable MPC sampling rates, and to do this a control horizon of 15 seconds is chosen.

In table 7.1 is a printout from the ACADO toolbox denoting the increasing problem complexity when the sampling time is decreased and the same control horizon is maintained. To see how the system react to this step with different scaled control horizons, a scenario is set up and plotted in figure 7.5. From this figure it can be seen what was expected above, when the sample time increases the control effort increases and because there's no viable constraint on the manipulated variable there's an immense overshoot when $N = 3$. This overshoot spreads throughout the system and almost all the pressure states reaches a 50% overshoot compared to the finishing value. Consequently this creates an overshoot also shown in figure 7.5. To solve the problem with the system breaching the physical constraint on the manipulated variable some constraints must be introduced to the optimization problem on the manipulated variable in section 7.5.

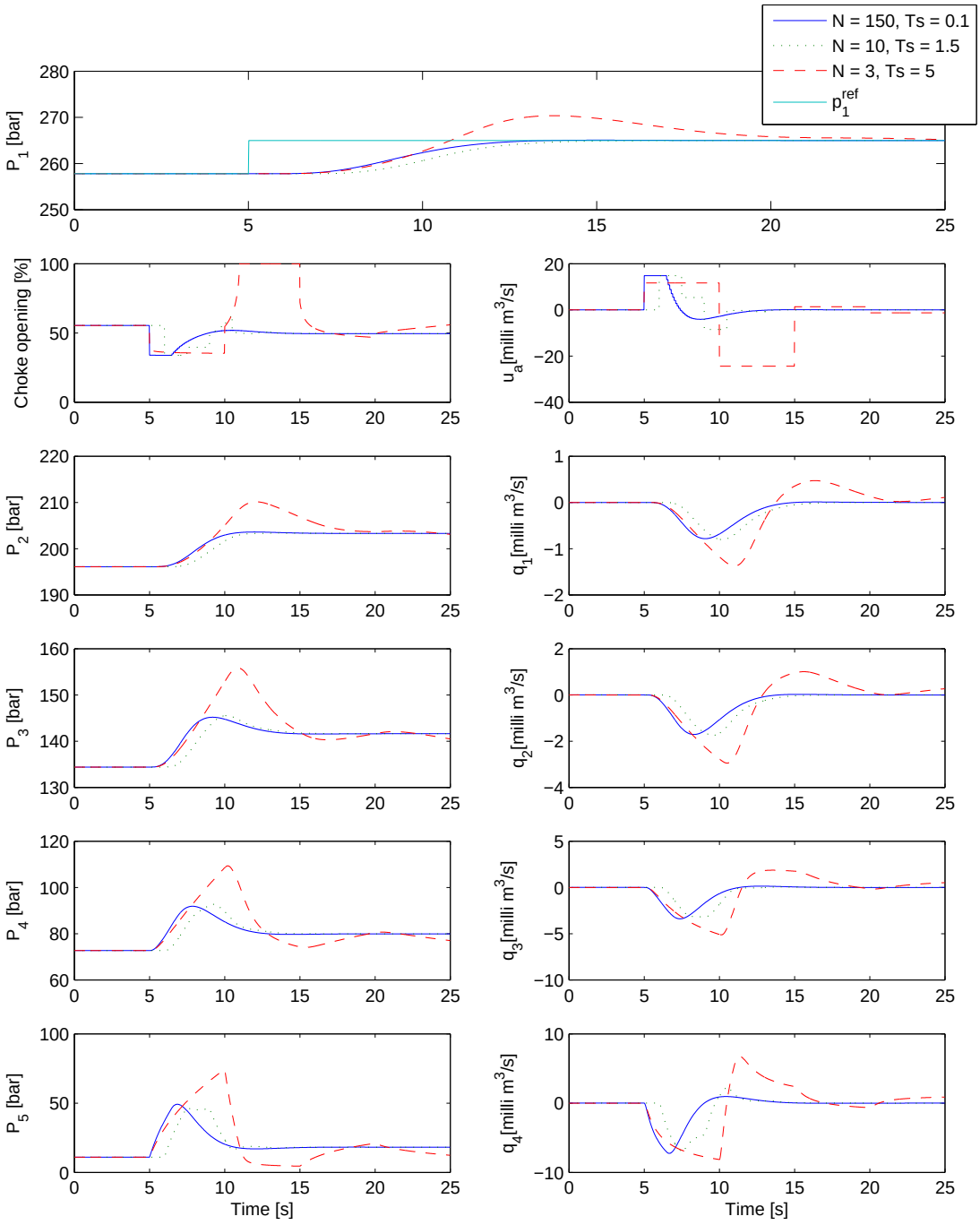


Figure 7.5: Estimated vs real disturbance

7.4 Sparse v.s. Condensed Qp

In section 4.6 two ways of formulation a quadratic MPC into a QP problem was presented, Sparse and the condensed implementation. It was discussed and concluded that it might be beneficial to choose the condensed method for solving a QP, if the problem was sufficiently small, depending on the structure of the problem. To get some exact data concerning computational complexity and memory consumption, the upper bound functions from table 4.1 are plotted for the MPD problem at hand to check whether the condensation is worth the effort. In a system with $m = 1$ manipulated inputs, $n = 9$ states and $l = 4$ constraints.

It might be useful to remember that these are upper bound functions and will not give us the exact solution but it will give us an idea where the border land is. And the figure tells us that in share processing power it will be profitable to choose a condensed solution if $N < 17$ and with respect to memory consumption it will be profitable if $N < 28$.

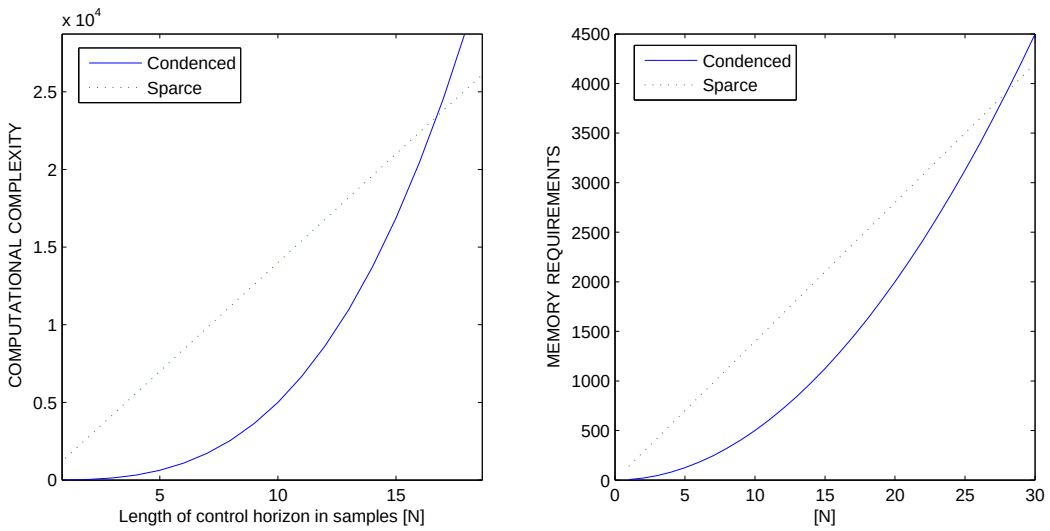


Figure 7.6: Computational time and Memory consumption

7.5 On-line constraint calculation

In section 7.3 the consequence of not having a constraint on the manipulated variable was introduced, and how this actually effects the convergence of the controlled variable. For the hole system the manipulated variable is the open and closed position of the choke valve, but because feedback linearization was used to cancel the non-linearities, the MPC manipulated becomes the volumetric flow rate u_a. This means the actual manipulated variable is calculated from u_a, which is dependent of squared pressure differential over the choke valve $\sqrt{p_c - p_0}$. To get an idea what to set the constraints to one can use the fact that the choke valve has a range between

$$0 \leq G(u) \leq 1$$

And the manipulated variable in the MPC is defined as

$$\begin{aligned} u_a &= q_{bpp} - q_c \\ &= q_{bpp} - K_c\sqrt{p_c - p_0}G(u) \\ &= q_{bpp} - K_c\sqrt{y - p_0}G(u) \end{aligned}$$

And because the real manipulated variable is the choke opening, the range of u_a can be defined by $G(u)$. Where the maximum volumetric flow rate $u_{a_{max}}$ occurs when $G(u) = 0$, and the lowest volumetric flow rate $u_{a_{min}}$ occurs when $G(u) = 1$. This gives the constraints.

$$\begin{aligned} u_{a_{max}} &= q_{bpp} \\ u_{a_{min}} &= q_{bpp} - K_c\sqrt{y - p_0} \end{aligned}$$

Which is easily implemented in the Matlab model by updating the constraint on each iteration. And for the ACADO implementation see section 6.5, where the constraints are adjusted from the state equations. Note that this will influence the problem complexity and table 7.1 migth not be viable for this setup.

Problems in the control horizon

The minimum manipulated variable constraint is a non-linear function of the topside pressure, and the topside pressure is varying over the control horizon. Because a liner MPC has been chosen the non-linearity can not be implemented in the optimization problem. To work around this problem the constraint $u_{a_{min}} = q_{bpp} - K_c\sqrt{y - p_0}$ can be set independently from the MPC at each sampling instant $[k]$. This of course mean that the constraint is valid at the time instant $[k]$ but it is not necessarily true for the rest of the control horizon $k + i$. This means that the constraint is only valid at the start of the control horizon and might cause damage into the horizon when it's not valid. An alternative to work around this problem is to only use the minimum constraint on the first step in the control horizon, which have been used in the rest of these simulations to improve the results although the MPC might then choose an optimal control path that isn't really a feasible option, and may actually exacerbate the results.

Problems whit low update rate

When the sampling time of the MPC T_{MPC} is high the MPC uses more extreme outputs to compensate for the lack of ability to change the output as we saw in section 7.3 where the control effort increased as a function of the sampling rate, which brings the manipulated variable closer to the constraints and increases the risk of a feasibility problem. This is brought up because when the MPC sets an output it stays the same until the next sample instant, and by this time the root $\sqrt{y - p0}$ may have changed so much that the physical constraint on p_c may be breached. This of course doesn't cause a risk of infeasibility in the MPC, and can to some extent be prohibited by increasing the sampling rate. This effect can be seen in figure 7.7 after 37 seconds when one can see that the MPC never brakes the constraints but the choke is passing 100% in the example with the lowest sampling rate. But in the example with the highest sampling rate, the constraint is never broken and the choke never exceeds 100%.

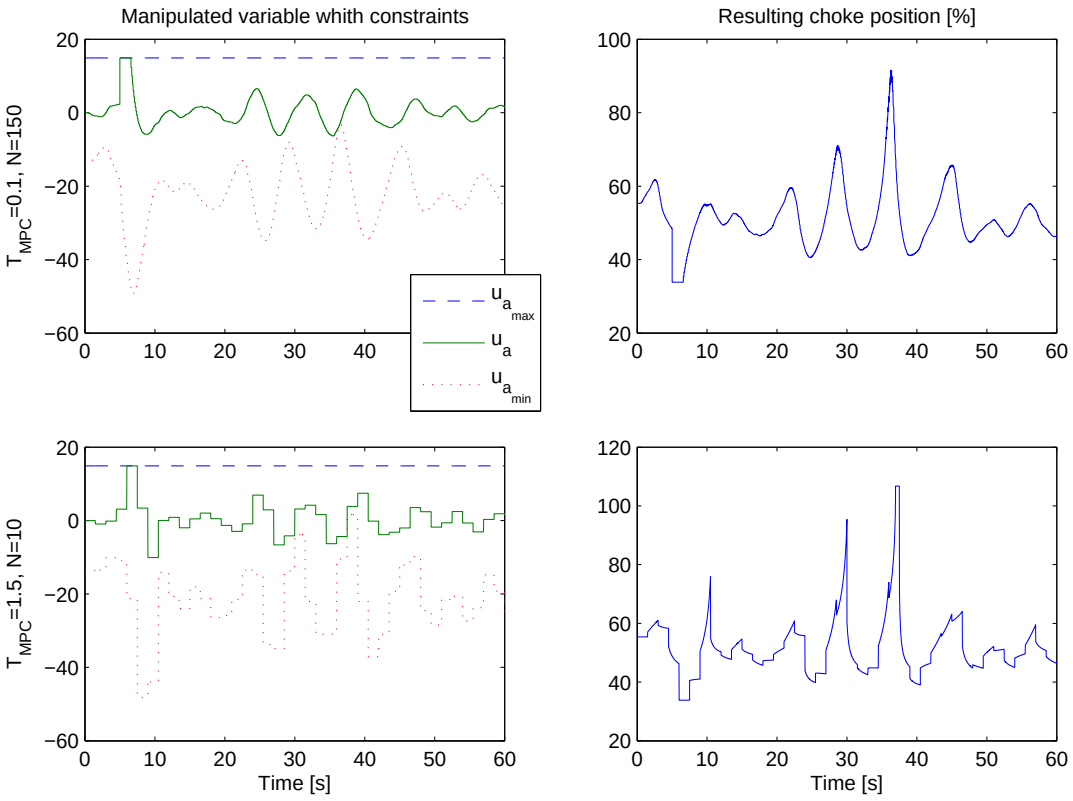


Figure 7.7: The manipulated variable with constraints and choke position

7.6 Disturbance Attenuation as a function of problem complexity

As shown in table 7.1, a MPC can be chosen with different sampling rates, and the QP complexity is determined by the sampling rate and the length of the control horizon. This means that the calculation power required to solve the QP problem is dependent on the sampling rate of the controller because we cant decrease the length of the horizon without risking the stability of the controller. Both the calculation power requirement and memory consumption is in principal important to keep low in all cases but because this controller is meant to be PLC implementable the requirement is even more vital. To see how the system with the disturbance presented in section 2.5 reacts with different MPC sample rates a test scenario is set up. In figure 7.8, one can see a step resonance of a condensed MPC without disturbance feedforward accumulated with different sampling times ranging from $T_{\text{MPC}} = 0.1$ to $T_{\text{MPC}} = 5$.

The first plot in this figure shows the controlled variable p_1 in the MPD system, and one can see that the disturbance rejection is pretty much unperturbed by the change of sampling frequency in the MPC. With that said, one can see from sub figure (2,2) that the control effort is considerably higher at lower sampling rates. Consequently the choke is oscillating at a higher frequency with a larger amplitude, which will cause the choke to move more and more rapidly. This will cause more wear and tear on the valve and shorten its durability and shorten it's life span. To make a stronger case for on how much worse the control effort becomes with decreased sampling rates we can summarize the control effort u_{a_t} over a time period.

$$\text{sum} = \sum_{t=20}^{60} = |u_{a_t}|$$

And to illustrate how much more control effort is needed for low frequency MPCs, the result is depicted in figure 7.9. Which shows that $T_{\text{MPC}} = 1.5$ is realistically probably the longest step length one can have before the overall system performance drastically decreases.

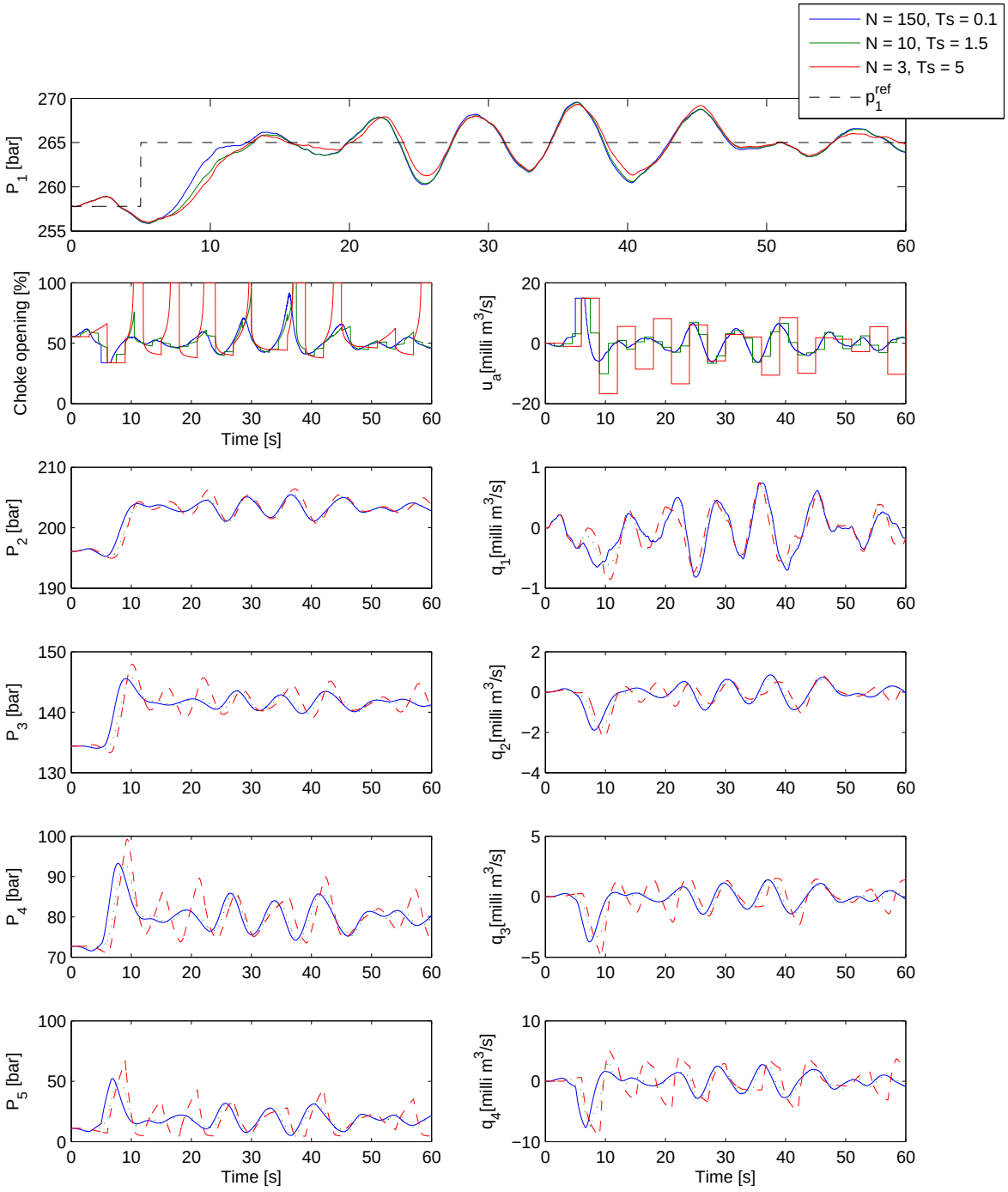


Figure 7.8: Step response with disturbances and different sample rates

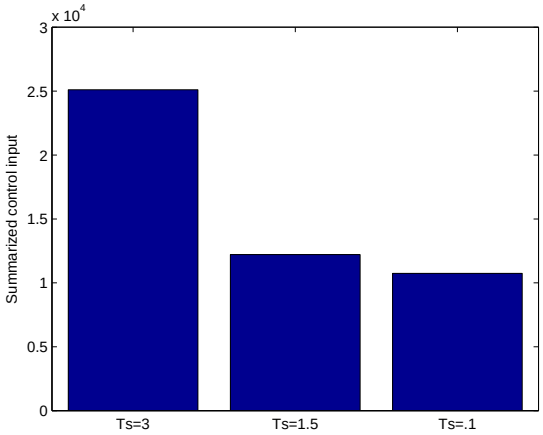


Figure 7.9: Summarized usage of u_a with different sample rates

7.7 FeedForward

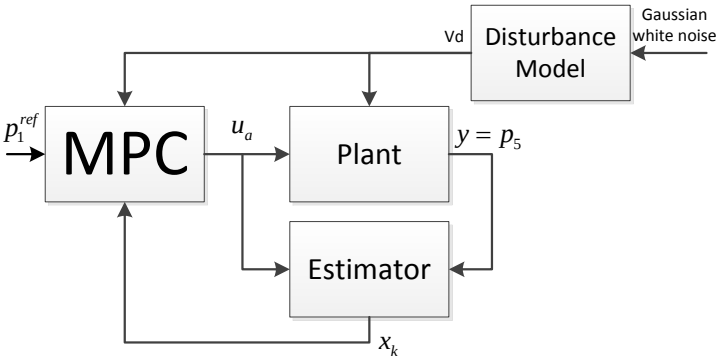


Figure 7.10: FeedForward block diagram

Feedback control systems have been around for a long time. The principle idea is to return the plant output, compare it to the reference and update the input to manage the error. A feedforward system on the other hand uses the knowledge of a disturbance or input to manipulate the output to prevent the disturbance to affect the system behaviour before an error occurs. This is a well documented and widely used technique which is often combined with a feedback control system. If both feedback and feadtforward are implemented there is an opportunity to both prevent disturbances from affecting the system and remove errors that occurs even with feedforward. Even if this is a common control principle it can be adopted into an optimal control problem by introducing the disturbance to the system equations. Figure 7.10 illustrates how the disturbance is distributed in the control system and how it can be counteracted by the MPC by introducing feedforward from the disturbance. Where the disturbance directly affects the down hull pressure and the manipulated variable u_a affect the topside pressure p_5. Which means that in order for the MPC to counteract the disturbance, it must manipulate the topside pressure in such a manner that it will spread to the down hole pressure and neutralize the pressure imposed by the disturbance. Although there is an obvious time difference from when an alteration on the manipulated variable is sett, until it reaches the down hole pressure. Another aspect of this is that disturbances is an unwanted system input, and unwanted influences is often unknown. This disturbance can actually be acquired from multiple sources, and in the following sections

one will see how well the overall system performance is affected by the different ways to acquire the disturbance information .

7.7.1 Measured rig motion

The drilling system is placed on a floating offshore installation presumable hold still by anchor or a DP system. If the installation is kept in place by a DP system or a DP system is installed, it would need detailed information of the ship motion to successfully keep the ship in position and it would certainly have accurate measurements of vessel movement in all directions. So its not a stretch to presume that the DP system have access to vessel velocity in heave direction. These measurements are measured by *Motion reference units*(MRU),*Vertical reference units*(VRU) or *Vertical reference units*(VRS) and are actually not limited to DP controlled vessels, which means there is a chance such devises is installed on a anchored vessel. Because our disturbance is caused by the ship velocity in heave direction this measurement can then directly be implemented as a feedforward in the MPC as shown in section 5.4. To show how the system will react and how much better the system performance would be with the measured heave velocity the system step response with the measured disturbance is depicted in figure 7.11. From this figure it seems like the disturbance rejection is much better than in the previous simulations but in reality the system disturbance has been stepped down from ±5 $[bar]$ in section 7.2, to ±1.5 $[bar]$ this will guarantee that the system never comes close to the constraint. The reason why we don't want the manipulated variable close to the constraint is because if the manipulated variable is using the whole span between the constraint, any potential improvement will be lost in the constraint. Furthermore the system converges fast to the reference, and by the movement of the choke there seems to some sort of disturbance rejection. This is done to guarantee that the constraints are never reached, as this could possibly corrupt the comparison.

7.7.2 Estimated rig motion

In the last section it was presumed that there were disturbance measurements available even though this is not granted. If you do not have access to disturbance measurements, a possibility is to estimate the disturbance. One way of gaining an estimated disturbance is by using a Kalman filter as simulated in section 7.1. To see how the system react to a feedforward control system where the disturbance is estimated the step response of the system is depicted in figure 7.11. This is clearly not an ideal solution but if the estimation is accurate it will do approximately the same job as a measurement as can be seen from the test scenarios of the kalman filter where the disturbance was depicted with the estimated disturbance in figure 7.3. It's easy to see that this is not an ideal solution, but it is pretty accurate as seen in section 7.1, and if it's not accurate the state estimates are not accurate either.

7.7.3 Measured rig motion with future predictions

We know that the MPC principle is to create a perfect control horizon, and to create a perfect control horizon with disturbance rejection it would be optimal to know the disturbance for the whole horizon. It is of course impossible to measure the future directly. But by measuring the waves on the ocean with radar or ultrasound technology it may be possible to predict the incoming waves before hand. If there then is a accurate RAO model of the drilling rig one could in principle predict the ship motion. The results of this is complete knowledge of the heave motions over the whole control horizon. To see how the system react to having a MPC with foresight to the disturbance on the whole control horizon a plot of the step resonance is shown in figure 7.11.

7.7.4 Comparison

In this section the scenarios from section 7.7.1-7.7.3 will be compared to tell how much feedforward improves the results, how much it matters where the disturbance measurement/estimate originates from and what information the estimates contains. To compare all the different aspects of the control problem in the comparison of the different aspects all the states and control inputs of the MPD system is plotted in figure 7.11. Sub plot (2,1) in this figure displays the choke opening for this for the different scenarios. From this figure one can see that for the systems with only feedforward from the present time gives approximately the same output, but if future information about the disturbance is introduced to the MPC controller the manipulated variable has a phase shift of almost 90 [deg]. This is of course not unexpected because the pressure travels at approximately the speed of sound and to contract a pressure wave coming 5 seconds from now it is reasonable that the manipulated variable has an offset. In the scenarios where the feedforward does not have future information the optimization looses the ability to counteract the disturbances. This is because when the disturbances occurs it is to late to contract them. The question then becomes how does this affect the performance of the down hole pressure and to depict this the sub plot 1 shows the down hole pressure. The problem with this plot is that it focuses on a too wide spectrum of pressures and the message does not quite come true. To get a closer look at the pressure we zoom in on the pressure between 20-50[s] in figure 7.12. This figure clearly shows that when future information is a part of the feedforward, the disturbance rejection gets better. The difference between the different sources of disturbances does not matter at all which confirms that the source of the disturbance does not matter. As discussed the measurement has no positive effect on the disturbance rejection because it comes to late to prevent any future disturbances. So if the well was 0 meeter long $p_1 = p_5$, there would not be a time delay and the disturbance could be removed completely by the manipulated variable without feedforward. Which implies that the importance of future predictions increases as the the well gets deeper, and dos not matter that much for a short well. This system have a zero mean disturbance but if there develops a bias on the disturbance the measurement feedback on the measurement would automatically counteract this bias from entering the system.

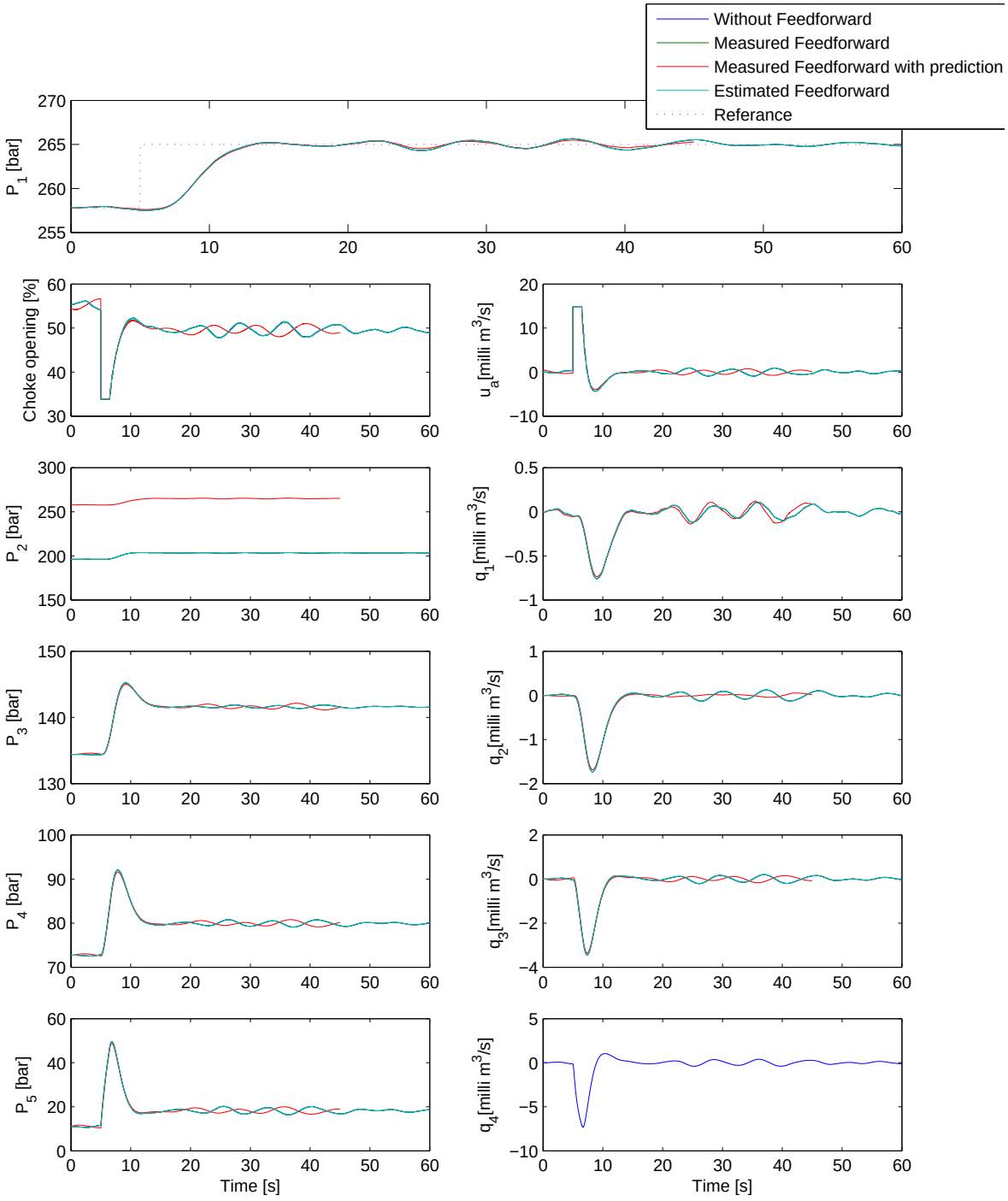


Figure 7.11: Comparison of disturbance rejection from different sources

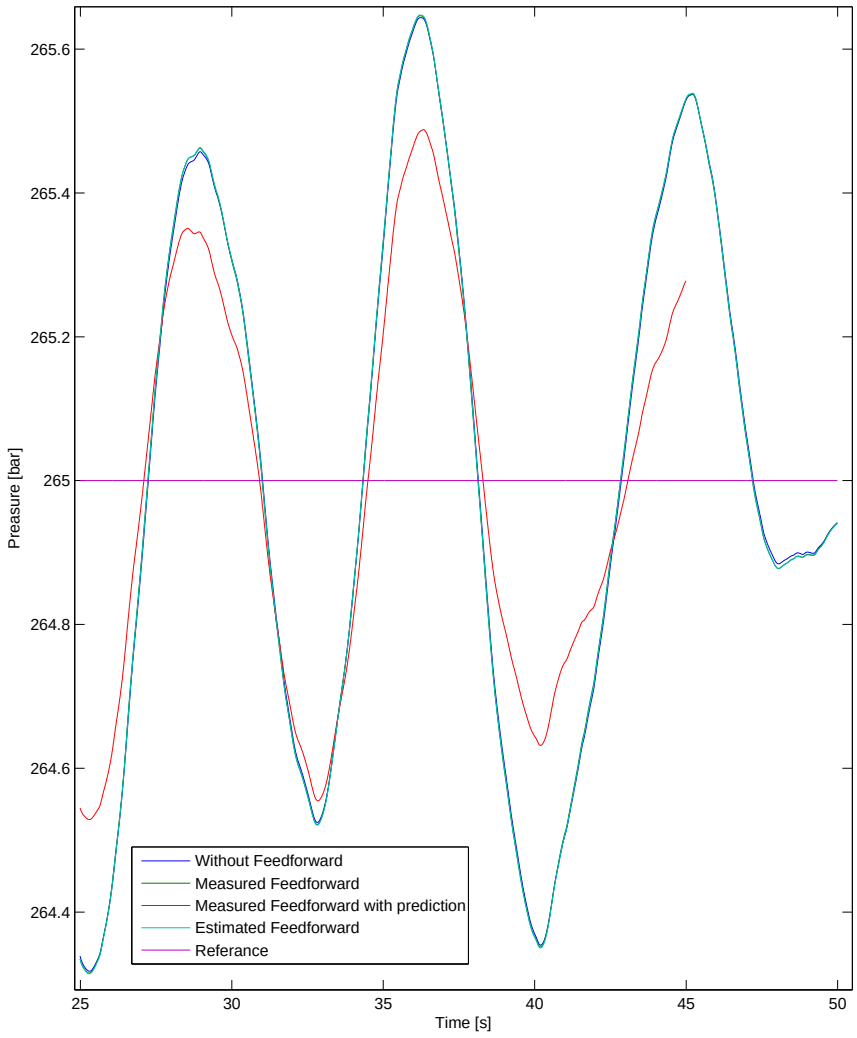


Figure 7.12: Comparison of disturbance rejection from different sources

7.8 PLC implementable MPC

When creating PLC implementable solution both memory availability and processing power requirement is immensely important if it's going to run properly, as discussed in section 7.8.2. To create an implementation as light as possible it's naturally to look at the comparison of update speeds from section 7.6. Where it clearly comes forward that selecting a solution faster then $T_{\text{MPC}} = 1.5$ will create a heavy wear on the choke valve and possibly not being feasible, so an update frequency of $T_{\text{MPC}} = 1.5$ is chosen for the implementation. The MPC problem is defined in a C++ script which is used to create a C solver solution. In this script it's possible to define variables to set in the runnable file. These variables can be used to implement the constraints from section 7.5 and one can use it to change the system equations online to fit the well depth. Because ACADO is made to make none-linear solvers it will naturally make a SQP solver, but if the problem is linear it will create an ordinary QP solver. One can also choose between different QP solvers and sparse or condensed implementations. For the simulations a sparse formulation is chosen with the Forces QP solver. The sparse formulation is chosen because it is written in C which is the supported language in the ABB AC500 PLC. The Kalman filter from section 7.1, is also implemented in the C script. The simulator and plotting function is implemented in Matlab and communicated over to the c program using matlab engine. For more detailed information about the implementation read chapter 6.

7.8.1 Simulation of PLC implementable MPC

To see how the C implementation of the MPC and kalman filter affects the simulations it is reasonable to plot the response of the system with disturbances. To best show the response of this control system all the states inputs and manipulated variables are depicted in figure 7.13. This figure clearly shows that the response of the ACADO implementation is identical to the response of the Matlab implementation in the sections 7.6. This validates the Matlab implementations above, from the perspective that the scenarios played out previously also is valid for the ACADO implementation. Further to analyse this scenario it might be helpful to notice that the previous simulations is 1 minute long, but this one is 5 minutes long. Sub plot (2,1) shows the choke valve is operating within the perimeters [0,100]% which shows that the constraints are upheld which

gives feasible sets, while converging nicely to the pressure references in sub plot 1.

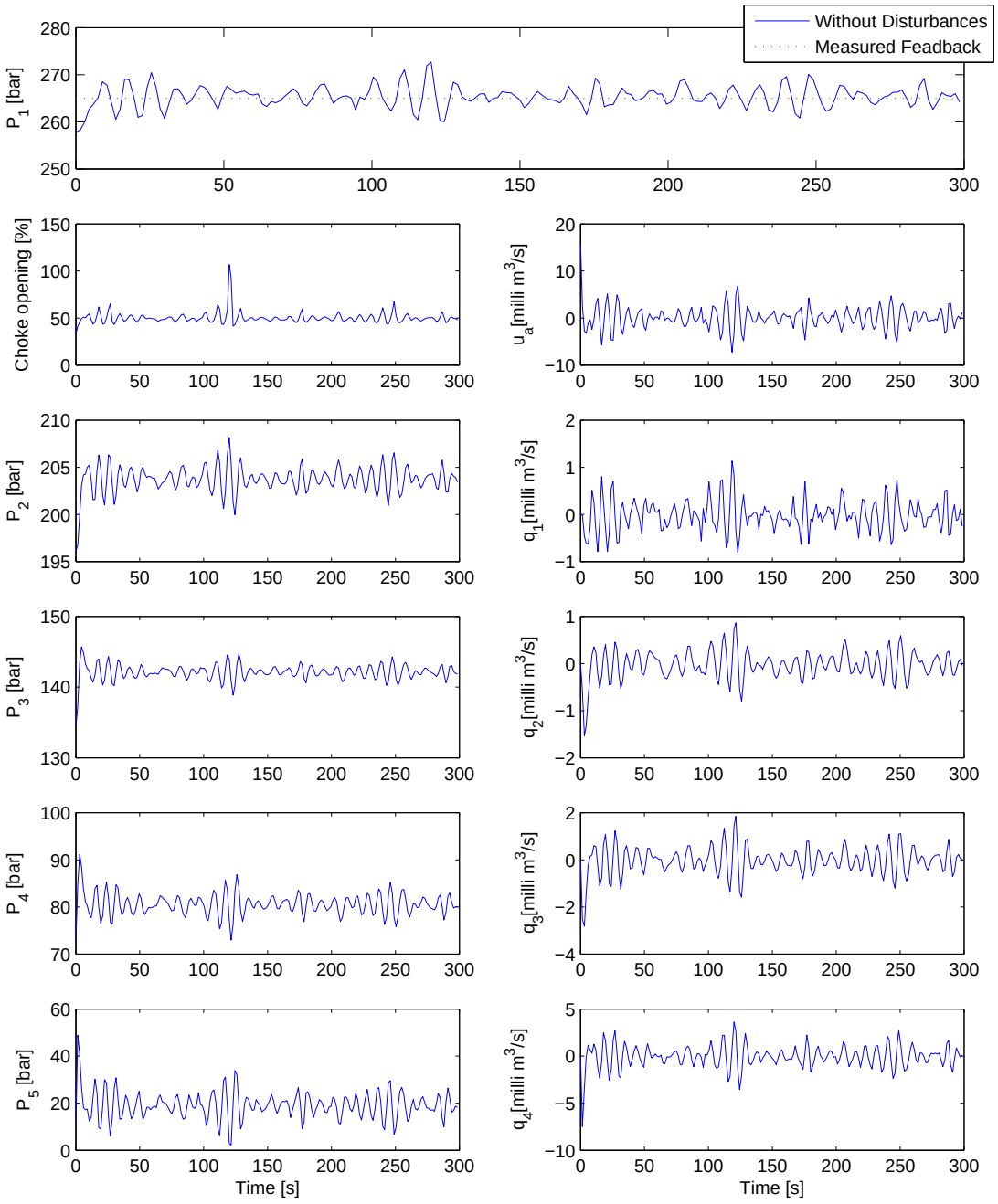


Figure 7.13: Acado Simulation

7.8.2 PLC preformance

The previous section now have been dedicated to engineer a suitable MPC controller for the sole purpose of being PLC implementable. There has also been designed an Kalman for state estimation and a process simulator to simulate the MPD system. In order to independently test the PLC MPC implantation from chapter 6 in a PLC, the simulator is moved from MATLAB to the C code implemented in the PLC. The reason why it is interesting to implement the MPC on an actual PLC is simple. To actually know if the designed MPC is runnable on the PLC. Where runnable it is meant that the memory usage is less than the available memory and the computation time is smaller than the time MPC sampling time. Optimally a condensed solver should be used, because this should deliver a better solver for MPC implementation on a PLC. But due to the lack of functionality in the PLC a sparse "Forces" solver was used. Which actually creates a better scenario if the results are satisfying, because if this scenario gives good results. The conceded solver will generate even better results. In time the QpOasis condensed solver will be available in C code, which will give even better results.

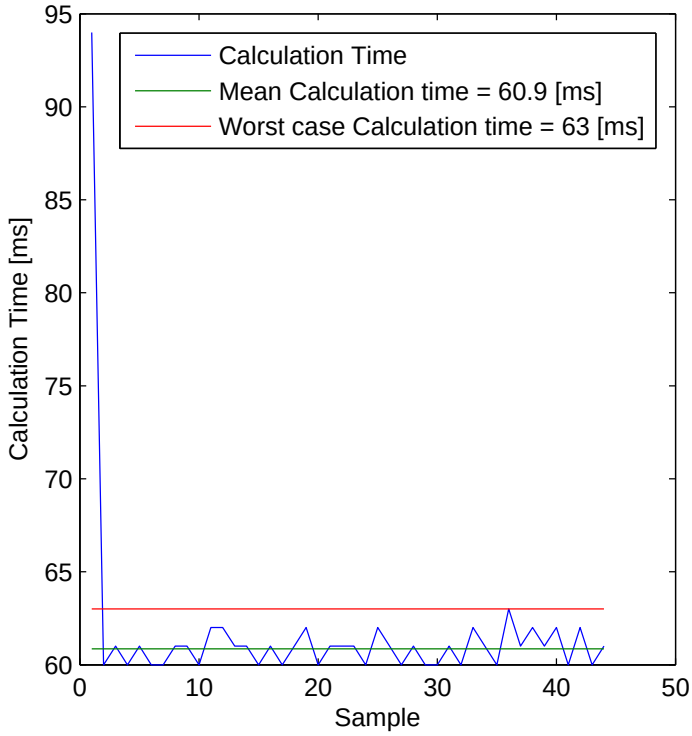
Memory consumption

PLC memory consumption is divided into two different sections, program code and program data. Program code are the memory allocated to the program code and program data is the memory allocated to data variables (int,double,Boolean,etc.). Figure 7.14(b) shows the memory consumption for the MPC implementation on the PLC. The code does not necessarily take more space if the problem is increased, while the number of variables will increase if the MPC complexity increases. Consequently the data consumption probably is the more critically of the two and only 2% is used.

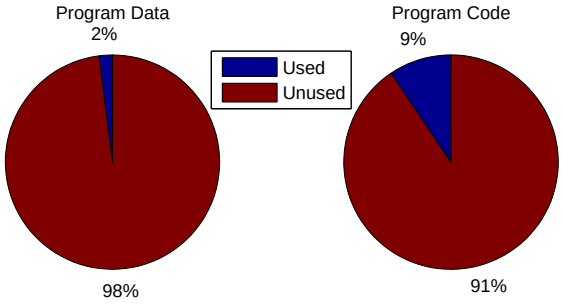
Calculation Time

As a simple rule when implementing a MPC controller is that the MPC update time should be at least twice as long as the computation time. The worst case computation time of the MPC shown in figure 7.14(a) is 63[ms] and the update speed of the MPC is $T_{\text{MPC}} = 1.5[s] = 1500[ms]$. Which means the PLC is 24 times faster than the MPC and we are well within the boundaries.

One might make an argument that because the PLC solves the problems so quick one might make the MPC faster. But it might also be useful to have large part of the PLC available because with this configuration it is possible to implement the MPC on a PLC already installed on the drilling rig, or use the PLC for multiple purposes.



(a) Computation time



(b) Memory consumption

Figure 7.14: PLC performance

Chapter 8

Discusion

State estimation

Because the state is not measurable a state estimator is implemented, or more specifically a Kalman filter to estimate the states. That automatically creates the question, is this estimator good enough or how good must it be? To answer these questions, you may want to look at what it will be used to. To use a MPC controller all the states must be presumed known, which they are not. And because of this a Kalman filter is employed to estimate the states. What makes the performance of this estimator so extremely important is that it estimates the controlled variable. And to make the estimation even harder, the disturbance affects the controlled variable directly. From the figures in section 7.1 one can see that the estimation error is worse on the controlled variable as a result of this. These simulations also shows that the estimation error becomes significantly better when one chooses a low sampling time, so how low should the sampling time be? The correct answer is probably as fast as possible. Because when the Kalman filter sampling time is $T_s = 1$, the estimation error becomes half the size of the disturbance it self. But this has limitations because choosing sample times faster than the sample time of the transmitter dos not give any improvements. So the sampling speed of the estimator should be chosen as fast as the sampling time of the topside transmitter. If one, for the sake of the argument, says that the minimum sampling time is $T_s = 1/16$. It can be seen from figure 7.1 that the maximum estimation error is under half a bar pressure. Which is only a fraction of the disturbance.

Non-linearity

In section 2.3 a dynamic model is presented, which is in large part linear with the exception of the choke valve equation.

$$q_c = \underbrace{K_c}_{\text{Constant}} \sqrt{\underbrace{p_c}_{\text{y}} - \underbrace{p_0}_{\text{Constant}}} \underbrace{G(u)}_{\text{Innput}} \tag{8.1}$$

This non-linearity of course makes the overall system non-linear. Now that a non-linear model has been presented, what alternative is there to implement optimal control? The solution is rather simple. One can either linearize the dynamical model and implement a linear MPC, or directly implement a non-linear MPC(NMPC). NMPC is a technology on the rise, and the problem could then be implemented directly. But there is some draw backs with this technology as with everything else. Because non-linear problems are in general harder to solve than linear problems, this solution would probably be less suitable for MPC implementation. Another aspect, is that ordinary MPCs are more established in the automation industry, and therefore easier to implement for possible users. So what must be done to implement a linear MPC, and what negative consequence does this present. First of, linearizing this model is possible. It can be done by using feedback linearization presented in section 2.5. The main drawback with this approach is that one dose not optimize the actual manipulated variable u but a new manipulated variable $u_a = q_{bpp} - q_c$ for the linearized system. But constraint on the manipulated variable is open / closed choke valve, can this constraint be used on the linearized MPC? In principle it can't be used, because if a non-linear constraint is added to a linear MPC it becomes a NMPC. So has our MPC become a NMPC or is there a way around this problem? In section 7.5 a solution to this problem was presented, where the constraints where calculated online. This approach had some drawbacks presented in section 7.5 but all in all this seemed to be a good solution, and for this reason it was used for the rest of the simulation.

Elaborating the control horizon

Because this system is meant to be PLC implementable it is important to choose a short as possible control horizon and this control horizon should be divided into as few samples as possible. If the sample time is large it will provide us with better time to calculate the optimal manipulated variable. The length of the control horizon on the other hand determines how complex

the problem will be, in the sense that there will be more optimal decision variables to calculate. This brings us the question of how much we can simplify the MPC without losing the virtues of the MPC controller? To even begin to respond to this question, one need to know how long time the system uses to reach steady state. This is because regardless of which method one choose to ensure stability from section 4.4, the performance, feasibility or stability is at stake. In section 7.3 it is shown that for most sampling frequencies a horizon of 15 seconds brings the states perfectly to steady state. Which again brings us to the question of how many optimization points are needed in the control horizon? To answer this one might look at what benefits many optimization points N gives. Section 7.6 tries to look at how much the sampling frequency helps on dampening the disturbance, but it concludes that the main effect of lowering the sampling frequency is that the control efforts becomes larger. So if the main drawback of having few optimization points is that the sampling time becomes high, which again will induce a higher control effort. In worst case this can create a feasibility problem on the manipulated variable, which will spread to the controlled variable to ensure feasibility. So how long can the sample time be before the control effort becomes significantly deteriorated? Figure 7.9 shows the summed control effort with different sampling times and it clearly depicts that the sampling time should not exceed $T_s = 1.5$ seconds.

Disturbance Attenuation

When reading section 7.6 one can see that the sampling frequency does not have any apparent effect on the disturbance rejection. This forms the question of why that is? We can start by looking at the main disturbance frequency $\omega_0 \approx 0.7222[rad/s] \approx 0.115$[Hz] which forms the Nyquist sampling theorem.

$$f_{\text{Nyquist}} > 2f \approx 2 \cdot 0.115 \approx 0.23[\text{Hz}]$$
$$T_{\text{Nyquist}} = \frac{1}{f_{\text{Nyquist}}} < 4.35[s]$$

And it can be seen that all of the sampling frequencies used in the simulation satisfies this disturbance and should have the ability to dampen the worst disturbances. So from a pure sampling frequency perspective all the frequencies should do an approximately equally good job remov-

ing the disturbance, which is the case. This brings us back to how good the disturbance rejection really is without feedforward. From figure 7.8 it is seen that the down hole pressure is oscillating from ±5 [bar] which is exactly the same as the disturbance in figure 7.4. This essentially tells us that the feedback MPC controller does not provide any disturbance rejection at all. Which should not be a surprise because the controller has no way of knowing about the disturbance, and asking the controller to fix a problem it know nothing about seems like a stretch.

To fix this problem it seems fitting to try giving the controller some knowledge about the disturbance and hope it will use it to counteract the disturbance. Which brings us to the feedforward simulations in section 7.7 where the results are depicted in figure 7.11-7.12. Which tells us that if feedforward is to be efficient it requires information about the disturbance in the whole control horizon. This is of course no surprise because it is impossible to counteract a future disturbance without future information. But it can still be useful to have feedforward without future information because it can prevent a potential steady state error on the down hole pressure if there becomes a bias on the disturbance. With that said the feedforward with future information dampens the disturbance with 25%, which is really impressive and way better than 0% which is the case for all the other scenarios. And to top it all off, it also provides the advantage of being able to prevent steady states errors caused by the disturbance.

PLC implementable solution

To implement the MPC controller on the PLC much of the information gathered above needs to be considered. Many of the questions posed earlier had the purpose of creating a PLC implementable solution as optimal for the job as possible. And as discussed when choosing the control horizon. Choosing a MPC which can be solved as easily as possible is essential, especial when making a controller which naturally requires a lot of computing power implementable on a device almost without uting power. And the question, how simple can the MPC controller be, is revisited. But this time the question is already answered. Because in section 7.6 it was found that the simplest justifiable MPC controller has a control horizon $N = 10$ and a sampling time $T_s = 1.5$. So how well does the PLC handle a MPC of this magnitude? From the simulations it actually looks like it handles it quite well, because the PLC only uses 1/24 of its disposable time.

That allows the PLC to actually have more computation time to spare for any other tasks it might be sensible to implement. But is this implementation as good as it appears in the Matlab simulations? To answer this question a simulation is set up to test the ACADO implementation in section 7.8.1 to see how well this implementation responds to the simulation with disturbances. And it appears to be operating exactly like the system in section 7.6.

Chapter 9

Conclusion

- **State estimation**

  The quality of the estimation is highly dependent on the sampling rate of the Kalman filter. This is because the disturbance is not completely random, and more samples gives more iterations to estimate the same disturbance. One of the main obstacles for the estimator was that the disturbance was directly influencing the controlled variable, which then becomes the hardest to estimate. In practical terms this would mean that one cold risk that the estimation error would be added on top of the disturbance, creating an even larger BHP variation. But with a sufficiently small sampling time the estimation error becomes negligible. The Kalman filter also seems to create a good estimate for the heave disturbance.

- **Optimization of the MPC for PLC implementation**

  From experimental simulations its found that the optimal control horizon is 15 seconds. This means that the problem complexity is a function of the sampling time in the MPC controller. From the simulations it is seen that the sampling speed in the MPC does not affect the disturbance attenuation, but it affect the control effort. The sampling time is therefore selected reasonably high without generating an unreasonably high control effort. $T_{\text{MPC}} = 1.5$

- **Condensation**

  If the control horizon for our problem is shorter than $N = 15$, a condensed approach is

preferable with respect to both computational time and memory consumption. ACADO does not support condensation for C code generation at this point, but a new QpOasis solver is in development which will provide this option. But this only reinforces the results of the PLC implementation, because we know it would be even better with this option.

- **Disturbance Attenuation**

  The MPC sampling time does not seem to have any effect on the disturbance attenuation if it is reasonably low. But if feedforward with future predictions is introduced the disturbance is damped with 25% at a depth of 2000 m. Even with this good result, the disturbance is still over pressure regulation accuracy, which would have been interesting on a more realistic drilling simulator or with some real life motion data from a drilling rig. The feedforward without future predictions dose not have any dampening effect on the disturbance but it could be used to prevent a bias to occur on the BHP if disturbance stopped being zero mean.

- **PLC implementation**

  The ACADO simulations lived up to the expectations in the sense that they where just like the Matlab simulations discussed above. The calculation time is 63ms which is 4.2 % of the MPC sampling time. Which is considerably better than 50 % which is generally a hand rule. The program data and code is also way within the limits with 2 and 9 percent respectively. Which would indicate that a much more software can be run on this simultaneously. It might even be possible to implement it on a PLC already installed on the drilling rig.

Appendix A

Condensed Qp

This chapter is going to formulate a condensed Qp white background from a sparse Qp formulation. The formulation is based on the condensed Qp in [ref til prosjekt]

Recursive model formulation

The system to be controlled is a discrete-time LTI system formulated on state space, as shown below.

$$x_{k+1} = Ax_k + Bu_k + Ev_{d_k} + B_{bias}$$
$$y_k = Cx_k$$

If one begins to recursively formulate the state space model forwards in time one can clearly see a pattern forming.

$$
\begin{aligned}
x_1 &= Ax_0 + Bu_0 + Ev_{d_0} + B_{bias} \\
x_2 &= Ax_1 + Bu_1 + Ev_{d_1} + B_{bias} \\
&= A(Ax_0 + Bu_0 + Ev_{d_0} + B_{bias}) + Bu_1 + Ev_{d_1} + B_{bias} \\
&= A^2x_0 + ABu_0 + AEv_{d_0} + AB_{bias} + Bu_1 + Ev_{d_1} + B_{bias} \\
x_3 &= Ax_2 + Bu_2 + Ev_{d_2} + B_{bias} \\
&= A(A^2x_0 + ABu_0 + AEv_{d_0} + AB_{bias} + Bu_1 + Ev_{d_1} + B_{bias}) + Bu_2 + Ev_{d_2} + B_{bias} \\
&= A^3x_0 + A^2Bu_0 + A^2Ev_{d_0} + A^2B_{bias} + ABu_1 + AEv_{d_1} + AB_{bias} + Bu_2 + Ev_{d_2} + B_{bias}
\end{aligned}
$$

This pattern makes it possible to reformulate the system matrices to a recursively state space valid for the control horizon.

$$
\underbrace{\begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \vdots \\ x_N \end{bmatrix}}_{x} = \underbrace{\begin{bmatrix} A \\ A^2 \\ A^3 \\ \vdots \\ A^N \end{bmatrix}}_{P_1} x_0 + \underbrace{\begin{bmatrix} B & 0 & 0 & \cdots & 0 \\ AB & B & 0 & \cdots & 0 \\ A^2B & AB & B & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}B & A^{N-2}B & A^{N-3}B & \cdots & B \end{bmatrix}}_{P_2} \underbrace{\begin{bmatrix} u_0 \\ u_1 \\ u_2 \\ \vdots \\ u_{N-1} \end{bmatrix}}_{u}
$$

$$
+ \underbrace{\begin{bmatrix} E & 0 & 0 & \cdots & 0 \\ AE & E & 0 & \cdots & 0 \\ A^2E & AE & E & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}E & A^{N-2}E & A^{N-3}E & \cdots & E \end{bmatrix}}_{P_3} \underbrace{\begin{bmatrix} v_{d_0} \\ v_{d_1} \\ v_{d_2} \\ \vdots \\ v_{d_{N-1}} \end{bmatrix}}_{v_d}
$$

$$
+ \underbrace{\begin{bmatrix} B_{bias} & 0 & 0 & \cdots & 0 \\ AB_{bias} & B_{bias} & 0 & \cdots & 0 \\ A^2B_{bias} & AB_{bias} & B_{bias} & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \\ A^{N-1}B_{bias} & A^{N-2}B_{bias} & A^{N-3}B_{bias} & \cdots & B_{bias} \end{bmatrix}}_{P_4}
$$

$$
\underbrace{\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_N \end{bmatrix}}_{y} = \underbrace{\begin{bmatrix} C & 0 & \cdots & 0 \\ 0 & C & \ddots & \vdots \\ \vdots & \ddots & C & 0 \\ 0 & \cdots & 0 & C \end{bmatrix}}_{P_5} \underbrace{\begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_N \end{bmatrix}}_{x}
$$

The output y can be defined as.

$$\begin{aligned} y &= P_5 x \\ &= P_5(P_1 x_0 + P_2 u + P_3 v_d + P_4) \\ &= P_5 P_1 x_0 + P_5 P_2 u + P_5 P_3 v_d + P_5 P_4 \end{aligned}$$

The objective function dose not necessarily contain all the states, but may instead weighted output variable. Or just the output as in this case.

$$\begin{aligned} J &= (y - y_{\text{ref}})^\top Q(y - y_{\text{ref}}) + u^\top R u \\ &= y^\top Q y - y^\top Q y_{ref} - y_{ref}^\top Q y + y_{ref}^\top Q y_{ref} + u^\top R u \end{aligned}$$

Because the cost function is scalar $y^\top Q y_{ref} = y_{ref}^\top Q y$, and the same goes for u/u_{ref}. Now all the terms which dos not contain decision can be removed, without this altering the solution.

$$J = y^\top Q y - 2 y_{ref}^\top Q y + u^\top R u$$

If we now set inn for y we get.

$$\begin{aligned} J &= (P_5 P_1 x_0 + P_5 P_2 u + P_5 P_3 v_d + P_5 P_4)^\top Q(P_5 P_1 x_0 + P_5 P_2 u + P_5 P_3 v_d + P_5 P_4) \\ &- 2 y_{ref}^\top Q(P_5 P_1 x_0 + P_5 P_2 u + P_5 P_3 v_d + P_5 P_4) + u^\top R u \\ &= x_0^\top P_1^\top P_5^\top Q P_5 P_1 x_0 + x_0^\top P_1^\top P_5^\top Q P_5 P_2 u + x_0^\top P_1^\top P_5^\top Q P_5 P_3 v_d + x_0^\top P_1^\top P_5^\top Q P_5 P_4 \\ &+ u^\top P_2^\top P_5^\top Q P_5 P_1 x_0 + u^\top P_2^\top P_5^\top Q P_5 P_2 u + u^\top P_2^\top P_5^\top Q P_5 P_3 v_d + u^\top P_2^\top P_5^\top Q P_5 P_4 \\ &+ v_d^\top P_3^\top P_5^\top Q P_5 P_1 x_0 + v_d^\top P_3^\top P_5^\top Q P_5 P_2 u + v_d^\top P_3^\top P_5^\top Q P_5 P_3 v_d + v_d^\top P_3^\top P_5^\top Q P_5 P_4 \\ &+ P_4^\top P_5^\top Q P_5 P_1 x_0 + P_4^\top P_5^\top Q P_5 P_2 u + P_4^\top P_5^\top Q P_5 P_3 v_d + P_4^\top P_5^\top Q P_5 P_4 \\ &- 2 y_{ref}^\top Q P_5 P_1 x_0 - 2 y_{ref}^\top Q P_5 P_2 u - 2 y_{ref}^\top Q P_5 P_3 v_d - 2 y_{ref}^\top Q P_5 P_3 v_d - 2 y_{ref}^\top Q P_5 P_4 \\ &+ u^\top R u \end{aligned}$$

And now the only decision variable is u, and the rest can be omitted without this altering the result, and each term is scalar an thus may be transposed.

$$J = x_0^\top P_1^\top P_5^\top Q P_5 P_2 u + u^\top P_2^\top P_5^\top Q P_5 P_1 x_0 + u^\top P_2^\top P_5^\top Q P_5 P_2 u + u^\top P_2^\top P_5^\top Q P_5 P_3 v_d$$
$$+ u^\top P_2^\top P_5^\top Q P_5 P_4 + v_d^\top P_3^\top P_5^\top Q P_5 P_2 u + P_4^\top P_5^\top Q P_5 P_2 u - 2 y_{ref}^\top Q P_5 P_2 u + u^\top R u$$

Thus it may be written.

$$J = u^\top \underbrace{(R + P_2^\top P_5^\top Q P_5 P_2)}_{H} u + \underbrace{2(x_0^\top P_1^\top P_5^\top + v_d^\top P_3^\top P_5^\top + P_4^\top P_5^\top - y_{ref}^\top) Q P_5 P_2}_{F\theta + f} u$$

beskrankninger

Where the constraints are defined.

$$u_{min,k} \leq u_k \leq u_{max,k}$$

$$y_{min,k} \leq u_k \leq y_{max,k}$$

Which can be written as.

$$\begin{aligned} u_k &\leq u_{max,k} & y_k &\leq y_{max,k} \\ -u_k &\leq -u_{min,k} & -y_k &\leq -y_{min,k} \end{aligned}$$

$$\underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{D_{u_k}} u_k \leq \underbrace{\begin{bmatrix} u_{max,k} \\ -u_{min,k} \end{bmatrix}}_{d_{u_k}} \quad \underbrace{\begin{bmatrix} 1 \\ -1 \end{bmatrix}}_{D_{y_k}} y_k \leq \underbrace{\begin{bmatrix} y_{max,k} \\ -y_{min,k} \end{bmatrix}}_{d_{y_k}}$$

Where the total constraint matrices is written as.

$$D_y y \leq d_y$$

$$D_u u \leq d_u$$

with

$$D_y = \begin{bmatrix} D_{y_1} & 0 & \cdots & 0 \\ 0 & D_{y_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & D_{y_N} \end{bmatrix}, \; D_u = \begin{bmatrix} D_{u_1} & 0 & \cdots & 0 \\ 0 & D_{u_2} & \ddots & \vdots \\ \vdots & \ddots & \ddots & 0 \\ 0 & \cdots & 0 & D_{u_N} \end{bmatrix}$$

$$d_y = \begin{bmatrix} d_{y_1} \\ d_{y_2} \\ \vdots \\ d_{y_N} \end{bmatrix}, \; d_u = \begin{bmatrix} d_{u_1} \\ d_{u_2} \\ \vdots \\ d_{u_N} \end{bmatrix}$$

Inserting the constraint model into the model yields:

$$\begin{aligned} D_y y &\leq d_y \\ D_y P_5 x &\leq d_y \\ D_y P_5 (P_1 x_0 + P_2 u + P_3 v_d + P_4) &\leq d_y \\ D_y P_5 P_2 u &\leq d_y - D_y P_5 P_1 x_0 - D_y P_5 P_3 v_d - D_y P_5 P_4 \end{aligned}$$

and

$$D_u u \leq d_u$$

Which gives rice to the constraints in a condensed QP formulation.

$$\underbrace{\begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix}}_{A_{ieq}} u \leq \underbrace{\begin{bmatrix} d_u \\ d_y - D_y P_5 P_1 x_0 - D_y P_5 P_3 v_d - D_y P_5 P_4 \end{bmatrix}}_{b_{ieq} + B_{ieq}\theta}$$

A.1 Condensed QP Formulation

And finally the MPC can be written on condensed QP form.

$$\begin{aligned} \min \quad & u^\top H u + (F\theta + f)u \\ \text{s.t.} \quad & A_{ieq} u \leq b_{ieq} + B_{ieq}\theta \end{aligned}$$

Where

$$\begin{aligned} H &= R + P_2^\top P_5^\top Q P_5 P_2 \\ F\theta + f &= 2(x_0^\top P_1^\top P_5^\top + v_d^\top P_3^\top P_5^\top + P_4^\top P_5^\top - y_{ref}^\top) Q P_5 P_2 \\ &= 2(P_1^\top P_5^\top Q P_5 P_2)^\top x_0 + 2(P_3^\top P_5^\top Q P_5 P_2)^\top v_d + 2P_4^\top P_5^\top Q P_5 P_2 - 2(Q P_5 P_2)^\top y_{ref} \\ A_{ieq} &= \begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix} \\ b_{ieq} + B_{ieq}\theta &= \begin{bmatrix} d_u \\ d_y - D_y P_5 P_1 x_0 - D_y P_5 P_3 v_d - D_y P_5 P_4 \end{bmatrix} \end{aligned}$$

Where θ is the QP initial contritions.

A.1.1 Reference Tracking MPC

Configuration of optimization problem with x_0 and the reference as initial conditions.

$$\begin{aligned} F &= \begin{bmatrix} 2(P_1^\top P_5^\top Q P_5 P_2)^\top & -2(Q P_5 P_2)^\top \end{bmatrix} \\ \theta &= \begin{bmatrix} x_0 \\ y_{ref} \end{bmatrix} \\ f &= 2(P_3^\top P_5^\top Q P_5 P_2)^\top v_d + 2P_4^\top P_5^\top Q P_5 P_2 \\ A_{ieq} &= \begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix} \\ b_{ieq} &= \begin{bmatrix} d_u \\ d_y - D_y P_5 P_3 v_d - D_y P_5 P_4 \end{bmatrix} \\ B_{ieq} &= \begin{bmatrix} 0 & 0 \\ -D_y P_5 P_1 & 0 \end{bmatrix} \end{aligned}$$

A.1.2 Reference Tracking MPC white disturbance feed forward

Configuration of optimization problem with x_0, reference and disturbance as initial conditions.

$$F = \begin{bmatrix} 2(P_1^\top P_5^\top Q P_5 P_2)^\top & -2(Q P_5 P_2)^\top & 2(P_3^\top P_5^\top Q P_5 P_2)^\top \end{bmatrix}$$

$$\theta = \begin{bmatrix} x_0 \\ y_{ref} \\ v_d \end{bmatrix}$$

$$f = 2P_4^\top P_5^\top Q P_5 P_2$$

$$A_{ieq} = \begin{bmatrix} D_u \\ D_y P_5 P_2 \end{bmatrix}$$

$$b_{ieq} = \begin{bmatrix} d_u \\ d_y - D_y P_5 P_4 \end{bmatrix}$$

$$B_{ieq} = \begin{bmatrix} 0 & 0 & 0 \\ -D_y P_5 P_1 & 0 & -D_y P_5 P_3 \end{bmatrix}$$

Appendix B

Adding a PID controller to plant model

This appendix contain a method to create a model that contain both the system and the low level controllers is the same model, as shown in figure B.1.

Simulator Model

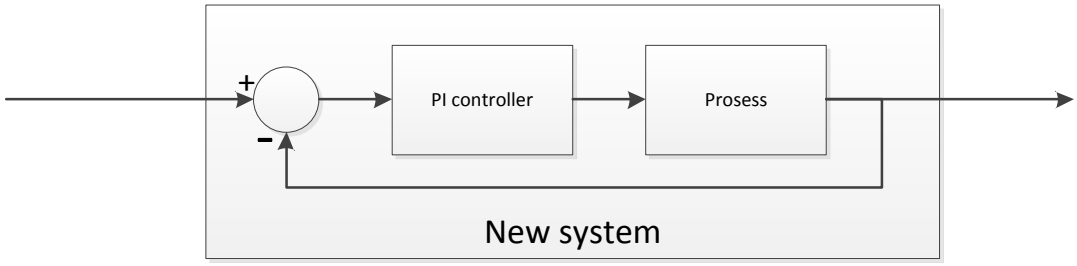


Figure B.1: PID master system

This new model shown in figure B.1, can be built from a range of different controllers, but due to it's popularity in industrial applications a PI controller was chosen. This section will describe how to create an new system model that includes both the controller and the physical system model. The PI controller is usually expressed on standard or parallel form but it can also

be written on state-space form as below.

$$\begin{aligned} \dot{x}_k &= B_k e \\ u &= x_k + D_k e \\ B_k &= T_i K_p \\ D_k &= K_p \end{aligned}$$

The controller above is then inserted into the process model from section[ref] and together they form the flowing model.

$$\dot{x}_p = A_p x_p + B_p(x_k + D_k e) + E_p v_d$$

This create the basis for a overall closed system loop system $x = [\dot{x}_p \;\; \dot{x}_k]^\top$ where $e = r - y$.

$$\begin{aligned} \dot{x} &= \begin{bmatrix} A_p x_p + B_p(x_k + D_k(r - C_p x_p)) + E_p v \\ B_k(r - C_p x_p) \end{bmatrix} \\ &= \begin{bmatrix} A_p x_p + B_p x_k + B_p D_k r - B_p D_k C_p x_p + E_p v \\ B_k r - B_k C_p x_p \end{bmatrix} \\ &= \begin{bmatrix} \begin{bmatrix} A_p - B_p D_k C_p & B_p \end{bmatrix} \begin{bmatrix} x_p & x_k \end{bmatrix}^\top + B_p D_k r + E_p v \\ \begin{bmatrix} -B_k C_p & 0 \end{bmatrix} \begin{bmatrix} x_p & x_k \end{bmatrix}^\top + B_k r \end{bmatrix} \\ &= \begin{bmatrix} A_p - B_p D_k C_p & B_p \\ -B_k C_p & 0 \end{bmatrix} \begin{bmatrix} x_p \\ x_k \end{bmatrix} + \begin{bmatrix} B_p D_k \\ B_k \end{bmatrix} r + \begin{bmatrix} E_p \\ 0 \end{bmatrix} v \\ &= A_{sim} x + B_{sim} r + E_{sim} v d \end{aligned}$$

Appendix C

Image Tutorials

C.1 Create New Project

This section is going to cover the basics of setting up a empty project in eclipse and creating a empty (C++) source file. The following enumerated list are built up in such a way that the number in the list represent the figure number.

1. **New project**

   Press the new project button to open the project manager faceplate.

2. **Select C++ project**

   Select new C++ project followed by Next.

3. **Define project**

   Select project name, empty project and select the compile of choice and finish by pressing Finish.

4. **New source file**

   To open the new source file faceplate left click on the project in the project explorer and select "New → Source File"

5. **Define source file**

   Select the name of the source file and finish by pressing Finish

Figure C.1: New project

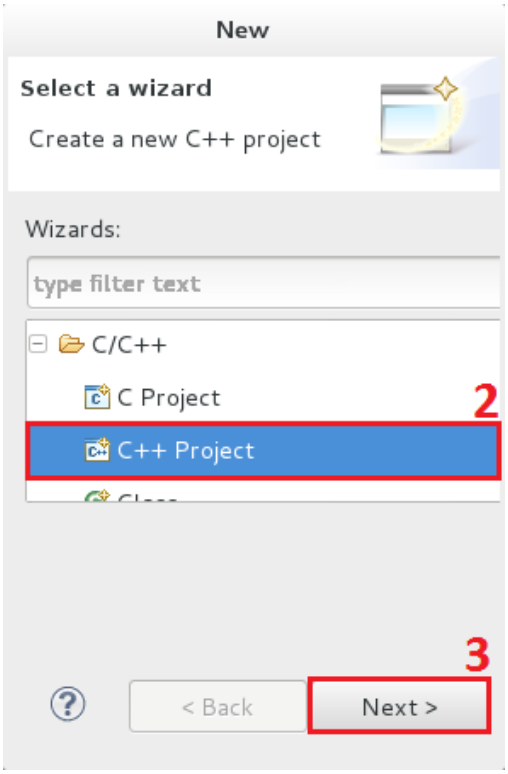


Figure C.2: Select C++ project

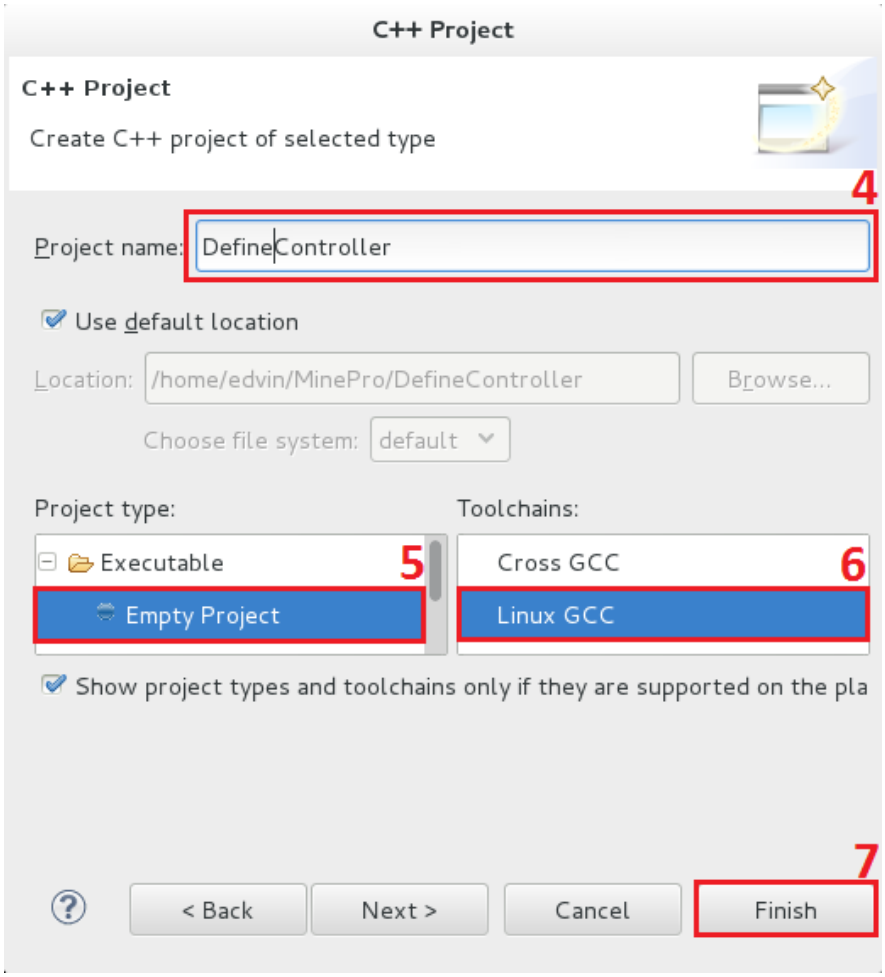


Figure C.3: Define project

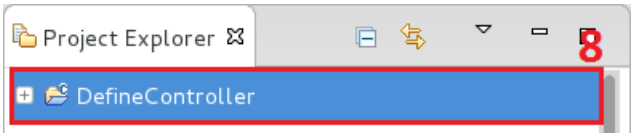


Figure C.4: New source file

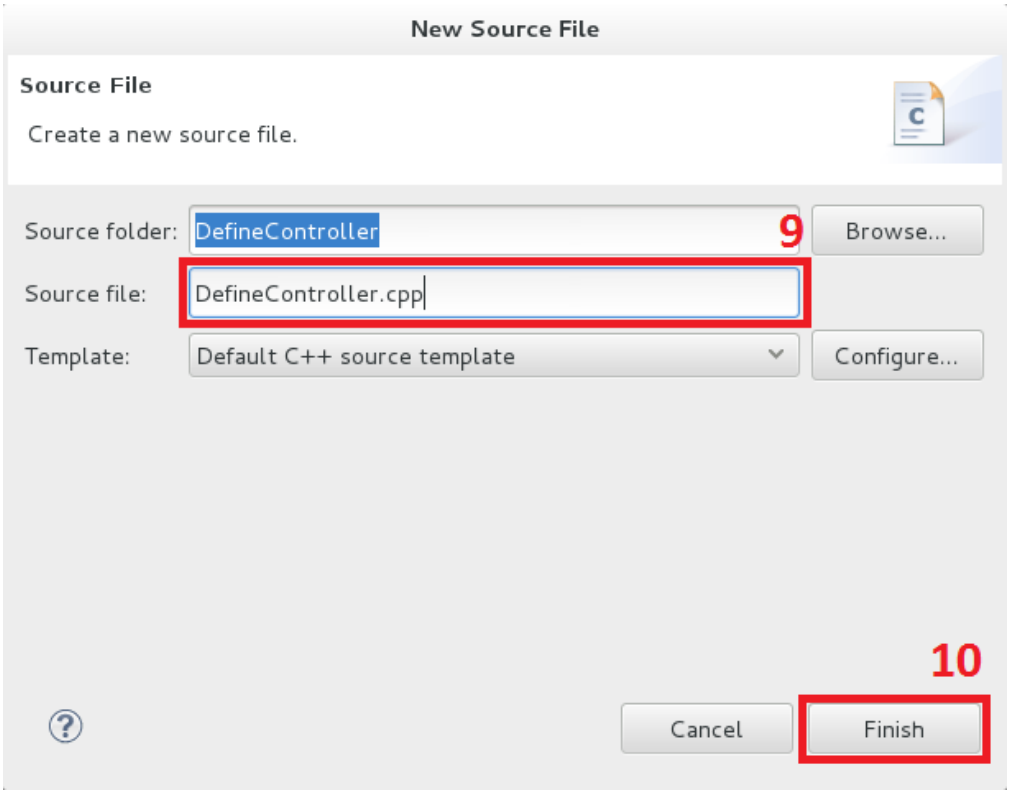


Figure C.5: Define source file

C.2 Setup project with makefile and executable

This section is going to graphically illustrate how to setep the project with the the makefile from CMake build managing software. And further make eclipse recognise the executable created the compilation defined in the makefile.

1. **Open Properties**

   To open the properties faceplate left click on the project in the project explorer and select "Properties"

2. **Change build options**

   Select "C/C++ Build" options in the tree on the left side. Deselect generate makefile automatically to stop eclipse from making a makefile then press the workspace button to select the pre made makefile from CMake.

3. **Select folder with makefile inside**

   Select the folder "build" inside the the project folder. If you don't have created this folder or the makefile within, the process is explained in section [ref]. When this file is selected press OK on both faceplate's to get back to the properties faceplate.

4. **Create new executable**

   The numbering jumps over one number her in-case you closed the properties faceplate, if you do go to point number one to open it again. Open the "Run/debug" in the tree on the left side, and press new build setting.

5. **Configuration type**

   Select that you want to build a C/C++ application and press OK.

6. **Executable configuration**

   Press the browse button to open a file browser, to select a executable from the file system. Note that if the code hasn't been compiled jet there is no executable file. To create a executable one can build the project or open the terminal maneuver to the build folder and type make.

7. **Select executable file** Maneuver your project folder select the executable from the make process and press OK until you are back to the editor.

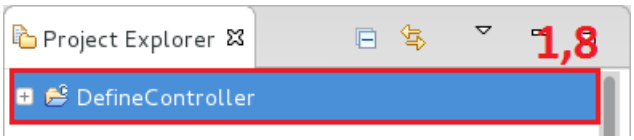


Figure C.6: Open Properties

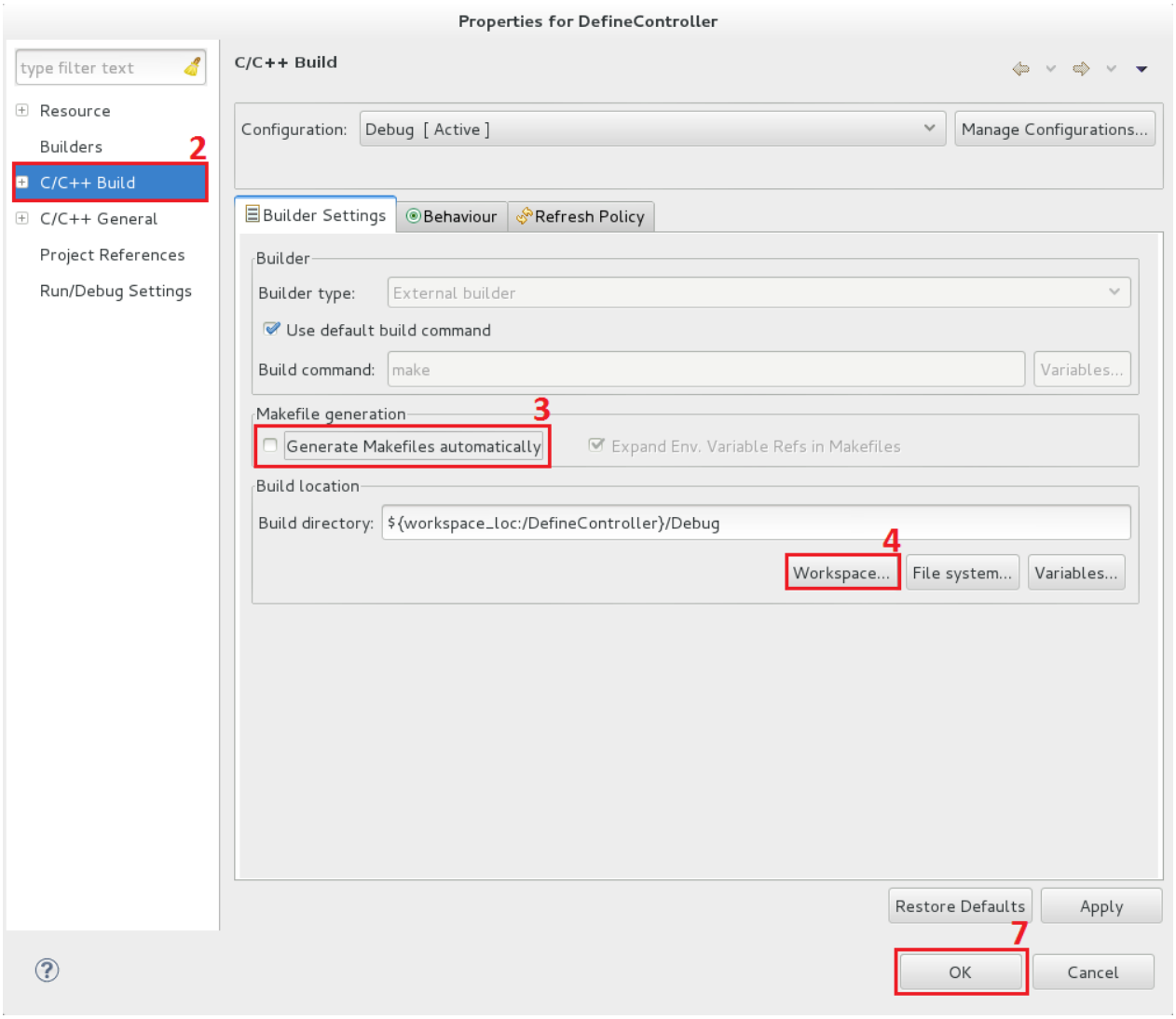


Figure C.7: Change build options

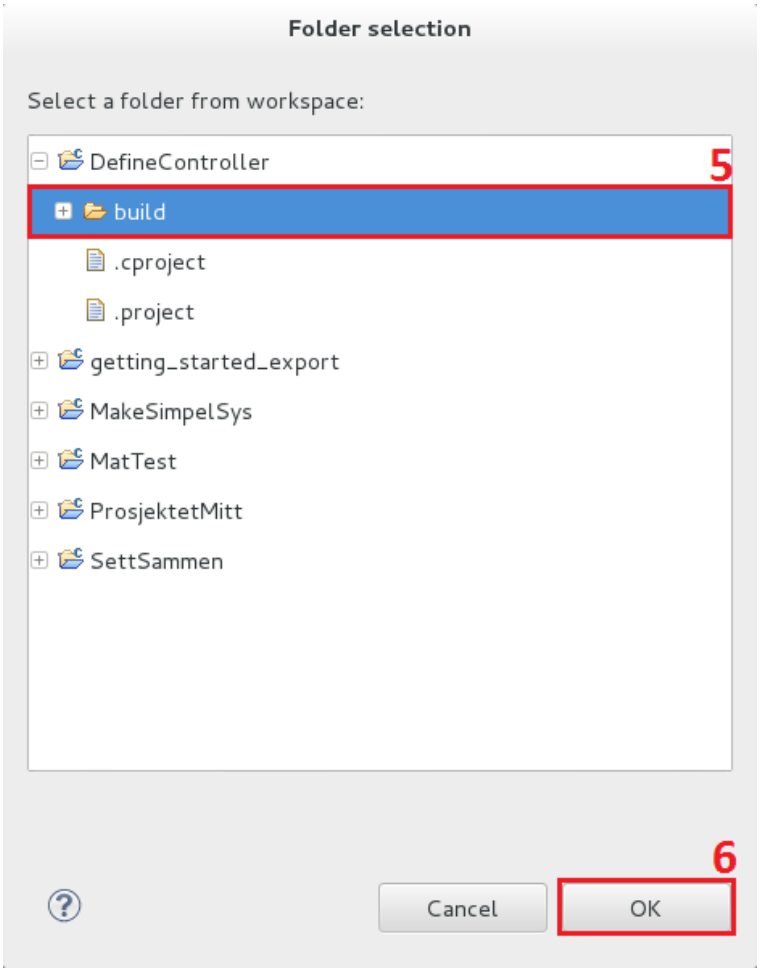


Figure C.8: Select folder with makefile inside

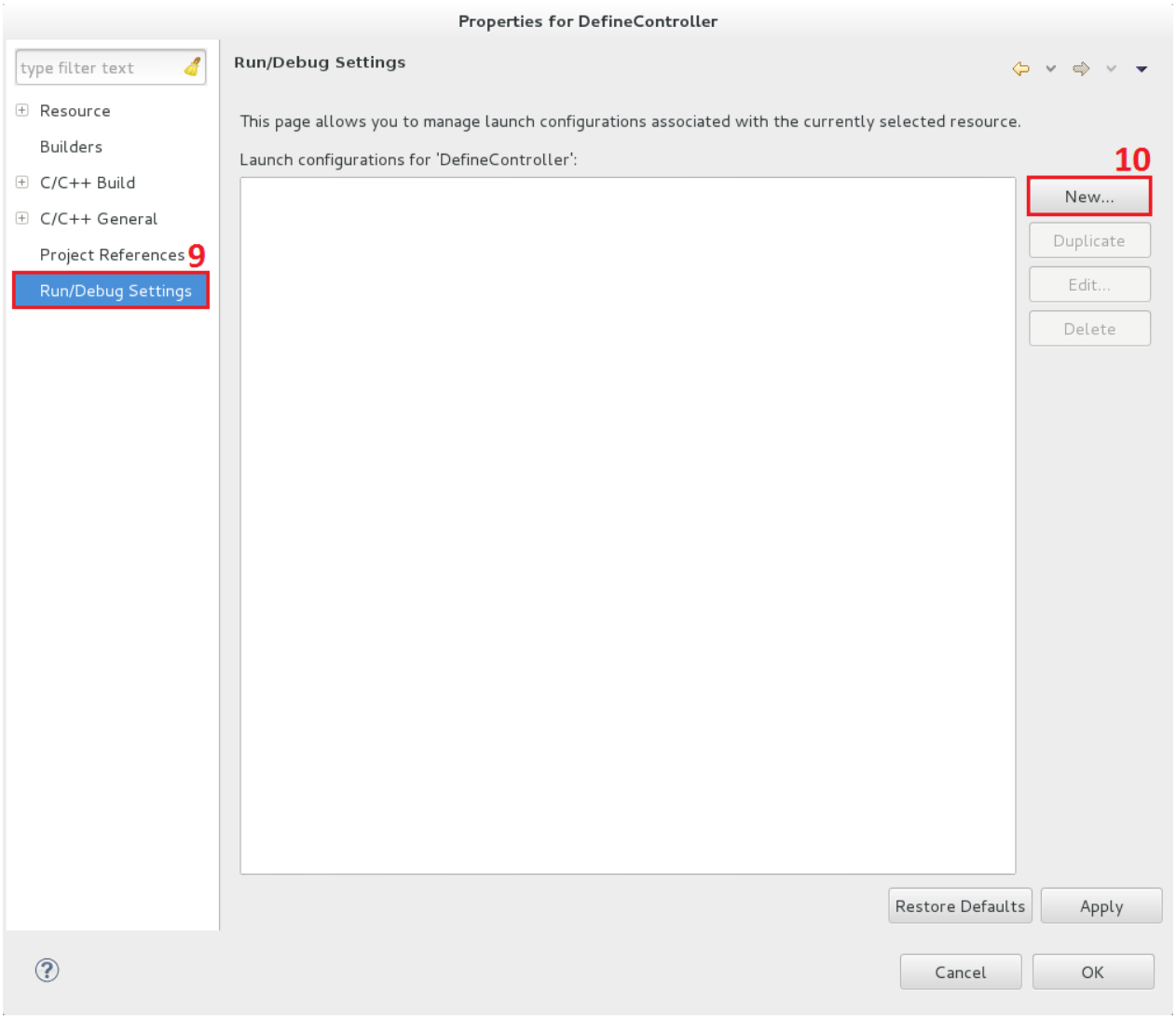


Figure C.9: Create new executable

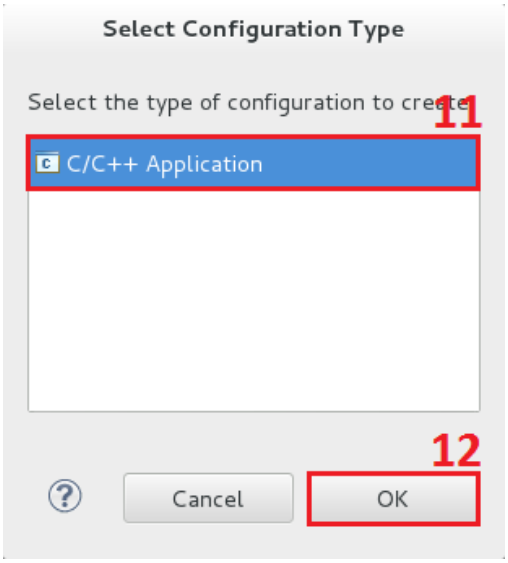


Figure C.10: Configuration type

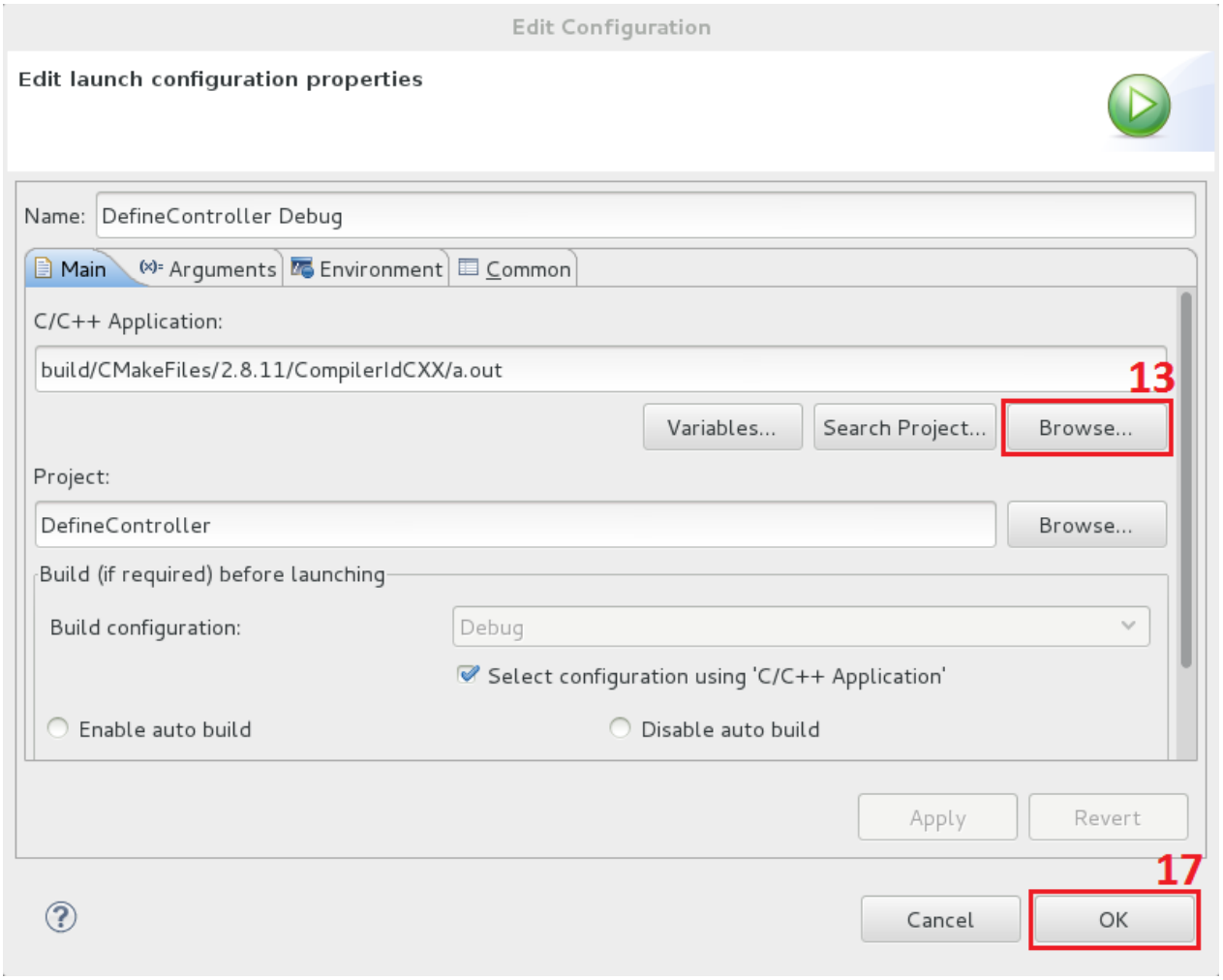


Figure C.11: Executable configuration

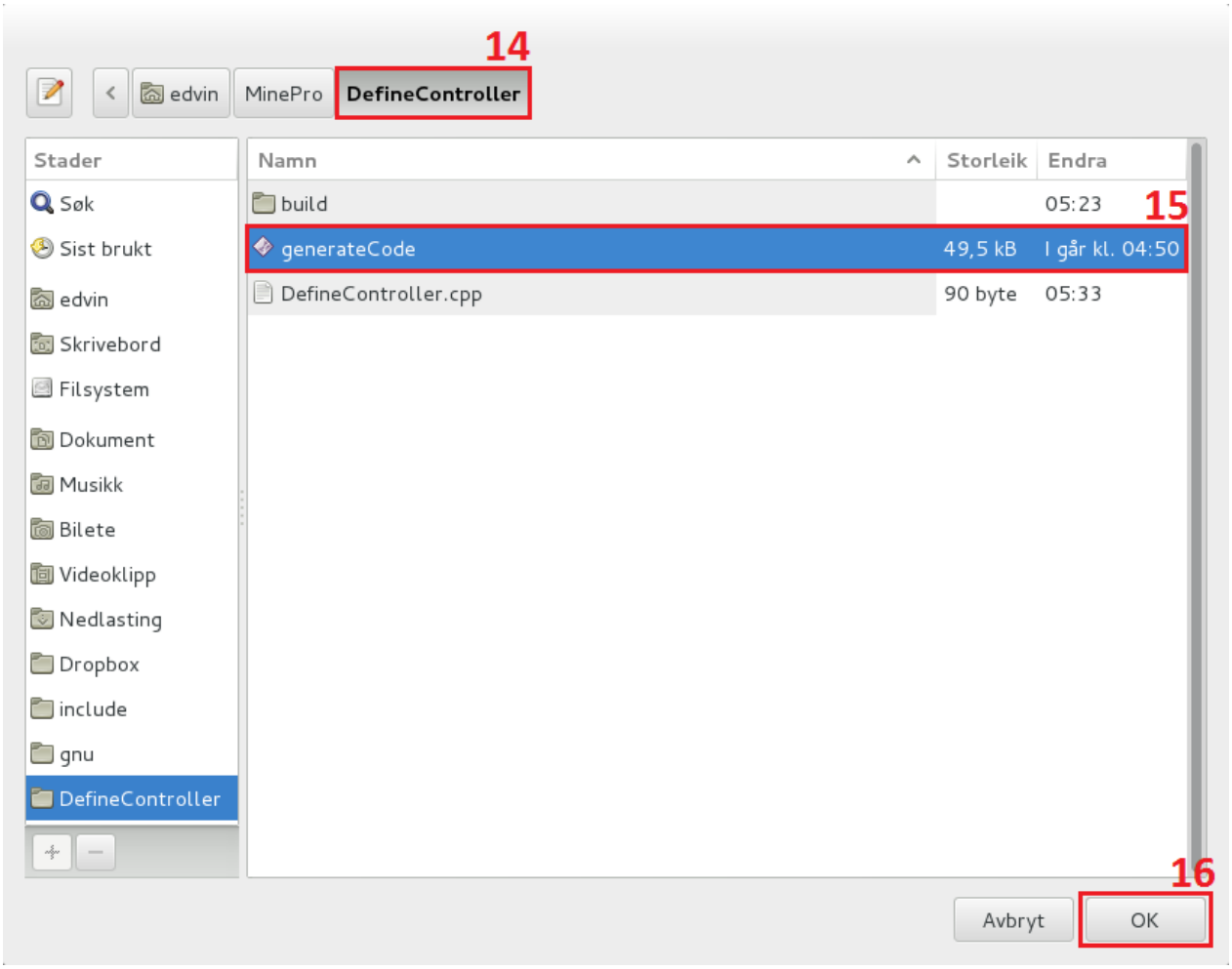


Figure C.12: Select executable file

Appendix D

Scripts

D.1 Code generation scripts

D.1.1 CMake settings File

To create a makefile for the code generation application in ACADO a build managing software called CMake is used. The build process setting is defined in a script. The settings file used in the MPD problem is denoted below.

```
# Minimum required version of cmake
CMAKE_MINIMUM_REQUIRED( VERSION 2.8 )

# Project name and programming languages used
PROJECT( MPD_MPC_SOLVER CXX )

# CMake module(s) path
SET( CMAKE_MODULE_PATH ${CMAKE_MODULE_PATH} ${PROJECT_SOURCE_DIR} )

#
# Prerequisites
#
FIND_PACKAGE( ACADO REQUIRED )

#
# Include directories
```

```
#
INCLUDE_DIRECTORIES( . ${ACADO_INCLUDE_DIRS} )


#
# Build an executable
#
ADD_EXECUTABLE( generateCode DefineController.cpp )
TARGET_LINK_LIBRARIES( generateCode ${ACADO_SHARED_LIBRARIES} )
SET_TARGET_PROPERTIES( generateCode PROPERTIES RUNTIME_OUTPUT_DIRECTORY
${CMAKE_CURRENT_SOURCE_DIR} )
```

Where "generateCode" is the name of the executable an "DefineController.cpp" is the name of the source code file.

D.1.2 Acado MPC for MPD script

Below is the complete script for the ACADO code generation. A background explanation and the meaning of the code lines is explained in section [ref].

```
#include <acado_toolkit.hpp>

int main( )
{
USING_NAMESPACE_ACADO

    // States:
DifferentialState   p_1;
DifferentialState   q_1;
DifferentialState   p_2;
DifferentialState   q_2;
DifferentialState   p_3;
DifferentialState   q_3;
DifferentialState   p_4;
DifferentialState   q_4;
DifferentialState   p_5;
DifferentialState   dummy;

    // Control inputs
Control u; // Control input
Control s; // Slcak variable

    // Disturbance
Parameter vd;

    // Constraints
Parameter uMin; // Disturbance
Parameter yMin; // Disturbance
Parameter yMax; // Disturbance

    // System Parameters
Parameter a;
```

```
Parameter b;
Parameter c;
Parameter e;
Parameter Kc;
Parameter p0;
Parameter qbpp;

    // Simulation Data
const double N = 10;  // Contol horizon
const double Ts = 1.5;  // Sampling time

    // Model equations:
DifferentialEquation f;
f << dot( p_1 ) == a*(-q_1 - vd*0.0656*21);
f << dot( q_1 ) == b*(p_1-p_2) - c*q_1-e;
f << dot( p_2 ) == a*(q_1-q_2);
f << dot( q_2 ) == b*(p_2-p_3) - c*q_2-e;
f << dot( p_3 ) == a*(q_2-q_3);
f << dot( q_3 ) == b*(p_3-p_4) - c*q_3-e;
f << dot( p_4 ) == a*(q_3-q_4);
f << dot( q_4 ) == b*(p_4-p_5) - c*q_4-e;
f << dot( p_4 ) == a*(q_4 + u);
f << dot(dummy) == s;

    // Reference functions and weighting matrices:
Function h, hN;
h << u << p_1; << s
hN << p_1;
Matrix W = eye( h.getDim() );
Matrix WN =  eye( hN.getDim() );
W(0,0) = 15;       // Input waigthing
W(1,1) = 150;      // Output waigthing
W(2,2) = 5000;     //  waigth of slack
WN(0,0) = 15;

    // Optimal Control Problem
```

```
OCP ocp(0.0, N*Ts, N);
ocp.subjectTo( f );
ocp.minimizeLSQ(W, h);
ocp.minimizeLSQEndTerm(WN, hN);

    // Constraints
ocp.subjectTo( p_1 + s - ymax <= 0);
ocp.subjectTo( 0 <= p_1 + s - ymax);
ocp.subjectTo(0 <= u - umin );
ocp.subjectTo( u -qbpp <= 0 );

    // Export the code:
OCPexport mpc( ocp );
mpc.set( HESSIAN_APPROXIMATION,       GAUSS_NEWTON    );
mpc.set( DISCRETIZATION_TYPE,         SINGLE_SHOOTING );
mpc.set( SPARSE_QP_SOLUTION,          SPARSE_SOLVER   );
mpc.set( INTEGRATOR_TYPE,             INT_RK4         );
mpc.set( NUM_INTEGRATOR_STEPS,        10              );
mpc.set( QP_SOLVER,                   QP_FORCES       );
mpc.set( GENERATE_TEST_FILE,          NO              );
mpc.set( GENERATE_MAKE_FILE,          NO              );
mpc.set( GENERATE_MATLAB_INTERFACE,   NO              );
mpc.set( GENERATE_SIMULINK_INTERFACE, NO              );

    return EXIT_SUCCESS;
}
```

D.2 Acado MPC simulator

D.2.1 Makefile

```
UNAME := $(shell uname)
MATLAB = /usr/local/MATLAB/R2013a/


LDLIBS = -lm -lstdc++
```

```
ifeq ($(UNAME), Linux)
        LDLIBS += -lrt -L/usr/local/MATLAB/R2013a/bin/glnxa64 -Xlinker -
           rpath -Xlinker /usr/local/MATLAB/R2013a/bin/glnxa64 -leng -
           lmx
endif
CFLAGS = -O3 -finline-functions -I. -I./qpoases -I$(MATLAB)/extern/
   include -I$(MATLAB)/extern/include/cpp
CXXFLAGS = -O3 -finline-functions -I. -I./qpoases -I./qpoases/INCLUDE -I
   ./qpoases/SRC -I$(MATLAB)/extern/include -I$(MATLAB)/extern/include/
   cpp
CC     = gcc
CXX    = g++

OBJECTS = \
        ./qpoases/SRC/Bounds.o \
        ./qpoases/SRC/Constraints.o \
        ./qpoases/SRC/CyclingManager.o \
        ./qpoases/SRC/Indexlist.o \
        ./qpoases/SRC/MessageHandling.o \
        ./qpoases/SRC/QProblem.o \
        ./qpoases/SRC/QProblemB.o \
        ./qpoases/SRC/SubjectTo.o \
        ./qpoases/SRC/Utils.o \
        ./qpoases/SRC/EXTRAS/SolutionAnalysis.o \
        acado_qpoases_interface.o \
        acado_integrator.o \
        acado_solver.o \
        acado_auxiliary_functions.o

.PHONY: all
all: libacado_exported_rti.a test

test: ${OBJECTS} test.o

acado_qpoases_interface.o    : acado_qpoases_interface.hpp
acado_solver.o               : acado_common.h
```

```
acado_integrator.o          : acado_common.h
acado_auxiliary_functions.o : acado_common.h \
                              acado_auxiliary_functions.h
test.o                      : acado_common.h \
                              acado_qpoases_interface.hpp \
                              acado_auxiliary_functions.h

libacado_exported_rti.a: ${OBJECTS}
        ar r $@ $?

${OBJECTS} : acado_qpoases_interface.hpp

.PHONY : clean
clean :
        -rm -f *.o *.a ./qpoases
```

D.2.2 The header file for the simulation

```
#include "acado_common.h"
#include "acado_auxiliary_functions.h"

#include <stdio.h>
#include <string.h>
#include <engine.h>

#define NX  ACADO_NX  // Number of differential state variables.
#define NXA ACADO_NXA // Number of algebraic variables. */
#define NU  ACADO_NU  // Number of control inputs.
#define NP  ACADO_NP  // Number of parameters.
#define NY  ACADO_NY  // Number of measurements/references on nodes 0..N
    - 1.
#define NYN ACADO_NYN // Number of measurements/references on node N.
#define N   ACADO_N   // Number of intervals in the horizon.
#define NUM_STEPS = 50  // Iterations to find optimal soution
int             SIM_STEPS = 500 // Length of simulation
```

```
/* Global variables used by the solver. */
ACADOvariables acadoVariables;
ACADOworkspace acadoWorkspace;

// Declare functions
int initMatlab();
void updateKalman();
int initAcado();
void acadoIter();

// Declare the system matrices
double Add[9][9] = // Discrerte system matrice Ad with Ts=1.5
double Bdd[9][9] = // Discrerte system matrice Bd with Ts=1.5
double Cdd[9] =    // Discrerte system matrice Ad with Ts=1.5
const double qbpp = 14.88;

// Define the kalman filter matrices
double Pm[10][10] = //The initial error covariance
double Q[10][10];
double xhat_m[10] = // inital state
double K[10],P[10][10];
double CPC,CXhatm,KC[10][10],KCT[10][10],PKC[10][10],PKCT[10][10],PA
    [10][10];
double xhat[10]
double xmat[10] = // inital state

// The input, output and disturbance.
double u[2];
double y;
double vd;

// Iterators for the system loop.
int    i,j, iter,step;

// Matlab enige variables
Engine *ep;
```

```
mxArray *mxVd;
mxArray *mxY;
mxArray *mxP5;
```

D.2.3 Initialisation of simulation environment

```
int initMatlab()
{
        ep = engOpen(NULL);             // Connect to MATLAB engine
        if(ep==0) {
                printf("Connecton to Matlab Engine failed\n");
                return(-1);
        }
        // Set a path to the matlab project folder and run th init
           script
        engEvalString(ep,"addpath('/home/edvin/Dropbox/Master/Matlab/
           cSimEin')");
        engEvalString(ep,"run('FungerendeParameterGull.m')");
        printf("Matlab Conection sucsess \n\n");
        return(0);
}

int initAcado(){
// Initialize the covariance matrix
Q[1][1] =       0.067472149950386;
Q[2][2] =       0.052295642557368;
Q[3][3] =       0.008542080531829;
Q[4][4] =       0.002259804065364;
Q[5][5] =       1.308157314771722e-04;

// Set the control bias
u[1] = 1;

// Clear solver memory.
memset(&acadoWorkspace, 0, sizeof( acadoWorkspace ));
memset(&acadoVariables, 0, sizeof( acadoVariables ));
```

```
// Initialize the solver.
initializeSolver();

// Initialize the states and controls.
for (i = 0; i < NX * (N + 1); ++i)  acadoVariables.x[ i ] = 0.0;
for (i = 0; i < NU * N; ++i)  acadoVariables.u[ i ] = 0.0;
// Sett the control referance
for (i = 0; i < N; ++i)
{
        for (j = 0; j < NY; ++j){
                if(j==0){
                        acadoVariables.y[i * NY + j] = 0;    // u
                }else{
                        acadoVariables.y[i * NY + j] = 265; // y
}}}
return(0);
}
```

D.2.4 Kalman filter implemented in C

```
void updateKalman()
{
        int i,j,k;
        // Equation for calculating K
        CPC = Pm[8][8];
        for(i = 0;i<9;++i){
                K[i] = Pm[i][8]/CPC;
        }
        // Equation for calculating /hat{x}
        CXhatm = Cdd[8]*xhat_m[8];
        for(i = 0;i<9;++i){
                xhat[i] = xhat_m[i] + K[i]*(y-CXhatm);
        }
        // Equation for calculating P
        KC[8][8] = KCT[8][8] = 1-Cdd[8]*K[8];
        for(i = 0;i<9;++i){
                KC[i][8] = -Cdd[8]*K[i];
```

```
            KC[i][i] = 1;
            KCT[8][i] = -Cdd[8]*K[i];
            KCT[i][i] = 1;
        }
        for (i = 0;i < 9; ++i){
             for (j = 0; j < 9; ++j){
                  PKCT[i][j]=0;
                   for(k = 0;k < 10; ++k){
                            PKCT[i][j] = PKCT[i][j]+Pm[i][k] * KCT[k][j];
        }}}
        for (i = 0;i < 9; ++i){
             for (j = 0; j < 9; ++j){
                  P[i][j]=0;
                   for(k = 0;k < 10; ++k){
                            P[i][j] = P[i][j]+KC[i][k] * PKCT[k][j];
        }}}
        // ------------------------------------------------------
        //               Preparer for next step
        // ------------------------------------------------------

        // Equation for calculating /hat{x}
        for(i = 0;i<9;++i){
            xhat_m[i] = 0;
            for(j = 0;j<9;++j){
                    xhat_m[i] += Add[i][j]*xhat[j];
            }
            for(j = 0;j<2;++j){
                    xhat_m[i] += Bdd[i][j]*u[j];
        }}
        // Equation for calculating P_m
        for (i = 0;i < 9; ++i){
             for (j = 0; j < 9; ++j){
                  PA[i][j]=0;
                   for(k = 0;k < 9; ++k){
                            PA[i][j] = PA[i][j]+P[i][k] * Add[k][j];
        }}}
```

```
        for (i = 0;i < 9; ++i){
             for (j = 0; j < 9; ++j){
                 Pm[i][j]=0;
                  for(k = 0;k < 9; ++k){
                          Pm[i][j] = Pm[i][j]+Add[i][k] * PA[k][j];
        }}}
        for ( i = 0 ; i < 9 ; ++i )
        // Q only has only element on the diagonal
               Pm[i][i] = Pm[i][i] + Q[i][i];
        vd =  xhat_m[0] - xhat[0];
}
```

D.2.5 Qp optimization

```
void acadoIter(){
preparationStep();
for(iter = 0; iter < NUM_STEPS; ++iter)
{
    // Perform the feedback step.
    feedbackStep( );

    // Chech if the KKT condition
    // satisfy a predefined reqierment
    if(getKKT()<0.0000001) break;

    // Preper for next step
    preparationStep();
}}
```

D.2.6 The Simulation loop

```
int main()
{
        // Run the init methods
        initMatlab();
        initAcado();
```

```
        for(step = 0; step < MPC_STEPS; ++step){
                // Get the disturbance from Matlab
                engEvalString(ep,"vvv = vd((i)*ceil(Ts/Tn));");
                mxVd = engGetVariable(ep, "vvv");
                acadoVariables.p[0] = (double)mxGetScalar(mxVd);
                // Set the intial condition to the optimization
                acadoVariables.x0[i] = xhat[i];
                // Run the optimization
                acadoIter();
                // Run the simulator one iteration forward
                u[0]  = acadoVariables.u[0];
                mxP5 = mxCreateDoubleScalar((double)u_a);
                engPutVariable(ep,"z",mxP5);
                engEvalString(ep,"[z] = UpdateMod(z,Add,Bdd,Edd,Cdd,vd((
                    i+1)*ceil(Ts/Tn)))");
                mxY = engGetVariable(ep, "z");
                y = mxGetScalar(mxY)
                // Estimate the states whith kalman filter.
                updateKalman();
        }
        // Save the results to a data file.
        engEvalString(ep,"save('/home/edvin/Dropbox/Master/Matlab/
            cSimEin/DistInfolessAcado.mat','x','y','u','vd');");
        engClose(ep);
        return(0);
}
```

Bibliography

(2014). *MATLAB® External Interfaces ,R2014a ,User manual*.

Amirhossein Nikoofard, Tor Arne Johansen, H. M. . and Pavlov, A. (2014). Design and comparison of constrained mpc with pid controller for heave disturbance attenuation in offshore managed pressure drilling systems. *Marine Technology Society Journal*, 48(2):90–103.

Chen, C.-T. (2009). *Linear System Theory & Design*. Oxford, 3st edition.

D. Q. Mayne, J. B. Rawlings, C. V. R. . P. O. M. S. (2000). Constrained model predictive control: Stability and optimalityq. *Automatica*, 36:789–814.

David Ariens, Moritz Diehl, H. J. F. B. H. F. L. R. Q. M. V. (2014). *Toolkit User's Manual*. acado-toolkit.org.

Erik Bølviken, N. C. S. (1998). *Linear dynamical models, Kalman filtering and statistics*. University of Oslo.

Fossen, T. I. (2011). *Marine Carft Hydrodynamics and Motion Control*. John Wiley & Sons.

Ægidius Mogensen, T. (2010). *Basics of Compiler Design*. lulu.com.

Godhavn, J.-M. (2010). Control requirements for automatic managed pressure drilling system. *SPE Drilling & Completion*, 25(03):336–345.

Heirung, B. F. . T. A. N. (2013). Merging optimization and control. *Lecture Note*.

Hhalil, H. K. (1996). *Nonlinear Systems*. Prentice-Hall, 3st edition.

Hovd, M. (2012). Lecture notes for the course advanced control of industrial processes.

Ingar Skyberg Landet, A. P. . O. M. A. (2013). Modeling and control of heave-induced pressure fluctuations in managed pressure drilling. *IEEE TRANSACTIONS ON CONTROL SYSTEMS TECHNOLOGY*, 21(4):1340–1351.

Johan Eck-Olsen, P.-J. P. . A. R. (2005). Maneaged pressures during underbalaced cementing by choking the return flow; innovative design and operational medeling as well as operational lessons. In *SPE/IDAC 92568*. SPE International.

Juan L. Jerez, E. C. K. . G. A. C. (2011). A condensed and sparse qp formulation for predictive control. In *50th IEEE Conference on Decision and Control and European Control Conference*.

K. Hasselman, T.P. Barnett, E. H. D. C. K. E. J. H. D. P. A. P. D. K. . W. H. (1973). Measurments of wind-wave groth and swell decay during the joint north sea wave project (jonswap). *Ergänzungsheft zur Deutschen Hydrographischen Zeitschrift*.

Kaasa, G.-O. (2012). Lecture in system identification. Un-publiched.

Marks, R. J. (1991). *Introduction to Shannon Sampling and Interpolation Theory*. Springer.

Mecklenburg, R. (2004). *Managing Projects with GNU Make*. O'reilly, 2st edition.

Møgster, J. (2013). Bruk av mpc for mpd. Master's thesis, NTNU.

Morari, A. B. . M. (1999). Robust model predictive control: A survey. *springer*, 245:207–226.

Olav Egeland, J. T. G. (2002). *ModeModel and Simulation for Automatic Control*. MARINE CYBERNETICS, 2st edition.

R.G. Brown, P. H. (2012). *Introduction to Random Signals and Applied Kalman Filtering*. John Wiley & Sons, 4st edition.

Skalle, P. (2011). *Pressure Control During Oil Well Drilling*. Ventus Publishing, 4st edition.

Wrigth, J. N. . S. J. (2006). *Numerical Optimization*. Springer, 3st edition.

Øyvind Breyholtz (2008). Nonlinear model predictive pressure control during drilling operations. Master's thesis, NTNU.

Øyvind Nistad Stamnes (2011). *Nonlinear Estimation with Applications to Drilling*. PhD thesis, Norwegian University of Science and Technology.