NTNU

Norwegian University of
Science and Technology

Development and Implementation of the Control System for an Autonomous Choke Valve for Downhole Flow and Pressure Control

Martin Øvsthus Christensen
Egil Gundersen

Master of Science in Cybernetics and Robotics
Submission date: June 2016
Supervisor: Ole Morten Aamo, ITK

Norwegian University of Science and Technology
Department of Engineering Cybernetics

Preface

This is the work of a 30 credits project which is the culmination of our masters degree in Cybernetics and Robotics at the Norwegian University of Science and Technology (NTNU). This thesis provides a description of the work that has been done and the results that were found during the spring semester of 2016. The objectives and the initial idea for this project is credited to Professor Ole Morten Aamo.

We would like to use this opportunity to thank our supervisor Professor Ole Morten Aamo for his guidance and inputs to this project. We would also like to thank Anders Rønning Dahlen for his work on acquiring the choke characteristic for the HeaveLock. Finally, we would like to thank our friend Tobias Tronstad Johansen for taking the time to review our thesis.

This thesis assumes that the reader has some experience in electronics, programming, general instrumentation and basic control theory.

Trondheim, 2016-06-06

Egil Gundersen

Martin Ø. Christensen

Abstract

Drilling operations in the oil and gas industry calls for tight control of well pressure. A drilling fluid is used to control the pressure in the well, remove cuttings, and lubricate and cool the drill bit. If the well pressure becomes either too low or too high, it can cause cave ins or fracturing of the well. When drilling operations are executed from a floating drilling rig, the drill string may move in and out of the well with the ocean waves. This will cause pressure fluctuations in the well. As of today, the downhole pressure is controlled from topside equipment, but due to the length of the drill string, significant delay makes pressure control difficult. A proposed solution to remedy this problem is to install a choke valve downhole, right above the drill bit. This choke valve and its control system has been named HeaveLock.

The aim of this thesis is to make the HeaveLock autonomous, and capable of controlling the downhole pressure in a lab at NTNU. The hardware and software that makes up the HeaveLock has been carefully selected and developed with autonomous operations in mind. The software that controls the HeaveLock is designed as a concurrent real-time program. It estimates the velocity of the HeaveLock based on acceleration, and uses the estimated velocity to control flow through the choke valve. By controlling the flow, pressure attenuation can be obtained. Both the velocity estimation and flow control algorithms are based on previous work, and has been discretized and adjusted to fit the purpose of the HeaveLock. A graphical user interface for a computer has been developed to enable effortless configuration and live data acquisition.

Every aspect of the HeaveLock has been tested both individually and collectively. The HeaveLock estimates velocity with good precision, and operates as intended. Unfortunately, due to the lab being occupied with other experiments, it was not possible to perform the final test in the lab within the deadline of this thesis. However, a suggestion for how such a test should be performed has been attached to this thesis.

With the exception of the pressure attenuation test of the HeaveLock in the lab, all objectives of this thesis have been met. The system is ready for the final test before being used in experiments.

Sammendrag

Ved gjennomføring av boreoperasjoner i olje- og gassindustrien er det viktig å regulere trykket til borevæsken. Borevæsken blir brukt for å transportere bort borekaks, regulere trykket i brønnen og for å kjøle og smøre borekronen i bunnen av brønnen. Dersom trykket i brønnen blir for lavt er det fare for at formasjonen kollapser og fanger borestrengen. Er det for høyt kan formasjonen sprekke opp, som kan føre til tap av borevæske inn i formasjonen. Under boring fra flytende borerigger vil bølger i enkelte situasjoner føre til at borestrengen beveger seg inn og ut av brønnen, som igjen fører til store trykkvariasjoner i brønnen. Tidligere har det vært forsøkt å regulere disse trykkvariasjonene ved bruk av utstyr på boreriggen, men grunnet lengden på borestrengen vil det oppstå en forsinkelse mellom pådrag og endring av trykket i bunnen av brønnen. Dette gjør reguleringen utfordrende. Et forslag til løsning har blitt lagt frem, som innebærer å installere en strupeventil like over borekronen i borestrengen. Strupeventilen med tilhørende styresystem har fått navnet HeaveLock.

Denne oppgaven tar for seg å gjøre HeaveLock autonom, og i stand til å regulere nedihullstrykket i et laboratorieoppsett ved NTNU. Maskinvare og programvare for HeaveLock har blitt valgt og utviklet for å muliggjøre autonom drift. Programvaren er et sanntidsprogram, med flere prioriterte oppgaver, som estimerer hastigheten til HeaveLock basert på akselerasjonsmålinger. Den estimerte hastigheten blir så brukt til å fastslå hvor stor strømningen skal være gjennom strupeventilen. Ved å regulere strømningen kan trykksvingningene i brønnen dempes. Hastighetsestimeringen og strømningsreguleringen er basert på tidligere arbeid, og er blitt diskretisert og tilpasset logikken for strupeventilen. I tillegg er det utviklet et grafisk brukergrensesnitt for en datamaskin, slik at sanntidsdata fra HeaveLock kan avleses og konfigurasjon er mulig uten å måtte endre programvaren.

Delene som utgjør programvaren og maskinvaren er testet både individuelt og som en total enhet. HeaveLock estimerer hastighet med god nøyaktighet, og er i stand til å operere som tiltenkt. Dessverre var det ikke tid til å utføre avsluttende tester i laboratoriet, grunnet annet arbeid som pågikk der. Det er derfor lagt ved en anbefaling for hvordan en slik test bør utføres.

Bortsett fra den endelige testen i laboratoriet er alle delmål oppfylt, og systemet er klart til siste test før enheten kan bli brukt til eksperimenter.

Acronyms

BHA Bottom Hole Assembly

CAL CAN Application Layer

CAN Controller Area Network

CAN-ID Controller Area Network Identifier

CBHP Constant Bottom Hole Pressure

CMSIS Cortex Microcontroller Software Interface Standard

COB-ID Communication Object Identifier

CoE CAN over EtherCAT

CPU Central Processing Unit

DC Direct Current

DSP Digital Signal Processing

EtherCAT Ethernet for Control Automation Technology

FAT File Allocation Table

FFT Fast Fourier Transform

FIR Finite Impulse Response

GND Ground (Electrical)

GUI Graphical User Interface

HLC HeaveLock Control

I^2C Inter-Integrated Circuit

IADC International Association of Drilling Contractors

IMU Inertial Measurement Unit

I/O Input / Output

IP Internet Protocol

IPT Department of Petroleum Engineering and Applied Geophysics (at NTNU)

MEMS Microelectromechanical System

MISO Master Input Slave Output

MOSI Master Output Slave Input

MPD Managed Pressure Drilling

NMT Network Management

OSI Open Systems Interconnection

PDO Process Data Object

P&ID Piping And Instrumentation Diagram

PID Proportional, Integral and Derivative

RMS Root Mean Square

RTOS Real-Time Operating System

Rx Receive

SCK Serial Clock

SDO Service Data Object

SIL Safety Integrity Level

SPI Serial Peripheral Interface

SS Slave Select

TCP Transmission Control Protocol

Tx Transmit

UART Universal Asynchronous Receiver/Transmitter

UDP User Datagram Protocol

USB Universal Serial Bus

Contents

Chapter 1

Introduction

1.1 Background and Motivation

Drilling of oil or gas wells require a drilling fluid called mud in order to remove cuttings, control pressure in the well, and cool and lubricate the drill bit. The mud circulates from the top of the drill string, through the drill bit and back to the drilling rig via the annulus of the well, bringing along the cuttings from the bottom of the well. To prevent influx of formation fluid such as oil and natural gas while drilling, the pressure in the well must be controlled to be above the pore pressure. A certain pressure must also be applied to prevent the formation from collapsing, trapping the drill string in the well. In addition to the lower pressure constraints, there are constraints to the maximum pressure. The pressure must be kept below the fracture pressure of the formation, to avoid damage to the reservoir and flow of drilling fluid into the formation. These limitations define the pressure window of the formation.

Traditionally the adjustment of well pressure has been done by changing the density of the drilling mud. In addition to hydrostatic pressure from the mud column, there will be an increase in pressure due to friction when the mud circulates. When the drill string has to be extended by connecting a new pipe, the main mud pump on the rig has to be shut down. As the flow is stopped, the pressure component caused by friction in annulus will be removed. This leads to large changes in well pressure, making it hard to drill in formations with a narrow pressure window. As this method has been used for many years, most of the “easy” prospects have already been drilled. Hence, many prospects that have been considered to be economically or techni-

cally un-drillable are left undeveloped. To be able to develop these prospects safely and within a reasonable budget, managed pressure drilling (MPD) is introduced.

The International Association of Drilling Contractors (IADC) have defined the concept of MPD to be "*an adaptive drilling process used to precisely control the annular pressure profile throughout the wellbore. The objectives are to ascertain the downhole pressure environment limits and to manage the annular hydraulic pressure profile accordingly [45].*" A variation of MPD called constant bottom hole pressure (CBHP) has been developed to overcome the challenges met when dealing with narrow pressure windows [21]. This method involves the use of a less dense drilling mud combined with a topside choke to increase the pressure in the well during circulation. The topside choke will only create an increased pressure as long as there is flow in the system, which is not the case during connections. To compensate for this, a back pressure pump is installed to create flow through the topside choke when the main pump is shut down. By controlling the choke and the flow of the back pressure pump, the pressure in the well can be kept steady at the desired setpoint. This technology has proven successful when drilling from stationary platforms.

When drilling operations are carried out from a floating drilling rig, a heave compensating system is usually used to counteract the heave motion of the drilling rig. During connections, this system has to be shut down to allow the drill string to be fastened to the drill floor. Heave motion of the floating rig will in this case move the drill string up and down like a piston in the well. This piston effect can induce large pressure fluctuations, causing the limitations of the formation pressure window to be exceeded. Several studies have been performed to find a method to compensate for these pressure fluctuations, most of which involves using some kind of topside equipment to control the pressure.

Due to the length of the drill string and annulus, there is typically a delay of a few seconds between a change in a topside choke opening and the resulting pressure change in the bottom of the well. To be able to compensate for wave disturbance completely, an accurate prediction of the heave motion of the rig over a few seconds would be necessary. This is however not possible in practice [42]. To solve the challenges of predicting heave motion, the actuator changing the pressure in the well can be moved closer to the bottom of the well. This is the idea of the Heave-Lock, a choke valve mounted in the bottom hole assembly (BHA) right above the drill bit, as seen

in Figure 1.1. This valve will control the flow of drilling mud during connections to compensate for the displacement of the drill string in the well.

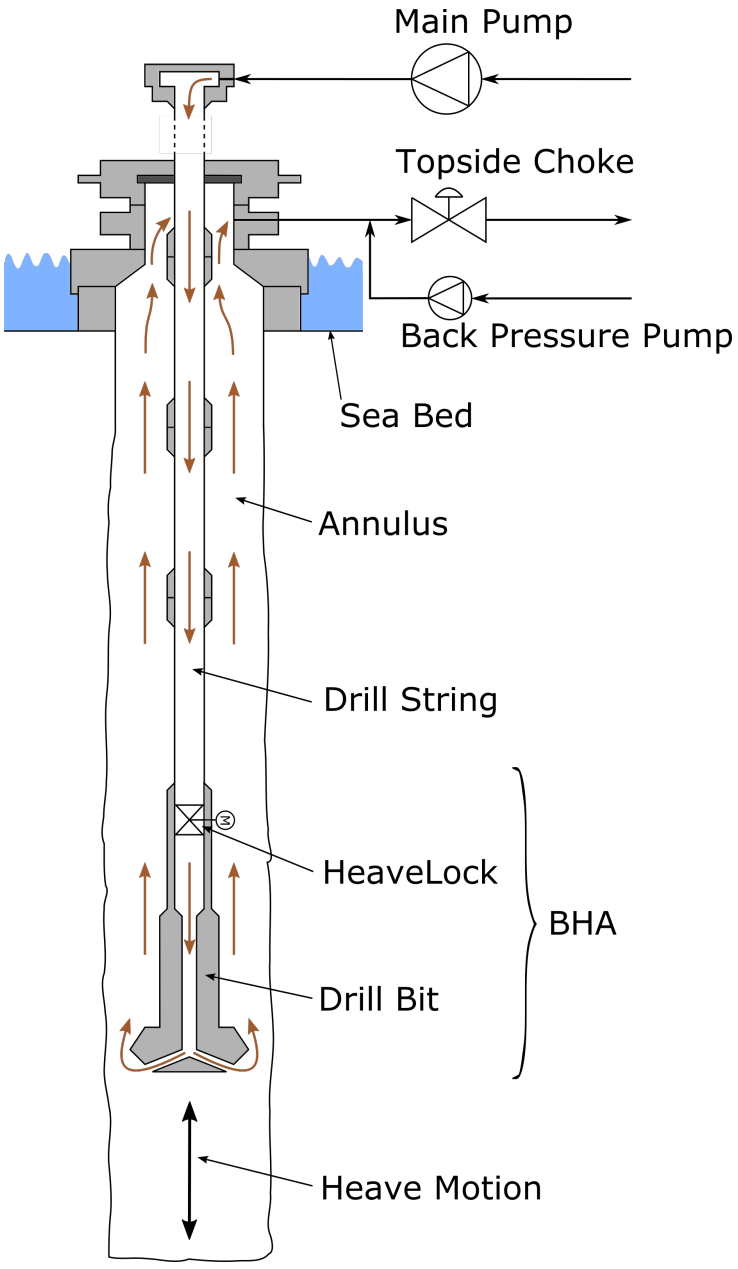

Figure 1.1: Overview of a drilling rig using MPD and the HeaveLock solution, adapted from [42].

1.2 Previous Work

At NTNU an experimental lab has been built (and rebuilt during the summer/fall of 2015) to model a scenario where the goal is to attenuate pressure fluctuations caused by a "drill string" moving in an annulus. The lab setup is called the Department of Petroleum Engineering and Applied Geophysics (IPT) Heave Lab, and is a scaled down version of a well with a drill string of approximately 900 m in length. The lab also includes a CBHP MPD system, and a system that simulates heave motion of the drill string. The piping and instrumentation diagram (P&ID) in Appendix A shows an overview of the lab.

In 2013 a system for generation of heave disturbance was developed by Drønnen [15], and a system using the topside choke to minimize the pressure fluctuations in the well was developed by Albert [1]. The best results obtained using this setup, with a model predictive controller, reduced pressure fluctuations caused by a heave disturbance with 3 seconds periods by approximately 46 %.

In order to test the principle of moving the choke valve downhole, a simulator of the system was created by Schaut [42] during the spring of 2015. The work of Schaut also includes suggestions to how the controller algorithms for the HeaveLock valve can be designed, and how the velocity of the BHA can be estimated using an accelerometer. The results from the simulations are quite promising, reducing the heave induced pressure fluctuations significantly.

During the fall of 2015 a design prototype for the HeaveLock valve was manufactured by IPT, and a position control system for the valve was developed by Gundersen [20] and Christensen [14]. The prototype used rotary solenoids to move the valve, and the lab control system sent valve position reference values to the position control system. Unfortunately this design failed. The differential pressure over the HeaveLock, in unison with a tight water seal, put to much strain on the rotary solenoids. As a result the valve did not move. In addition, the cross sectional flow area of the valve was too large, leading to too low differential pressure over the valve, making it hard to control the flow. This required a redesign of the HeaveLock valve. During the spring of 2016 a new HeaveLock valve was manufactured by IPT, shown in Figures 1.2 and 1.3.

As a step towards making the HeaveLock operate autonomously, it was decided that the new HeaveLock unit should incorporate much of the logic in the control system. In a real well, com-

Figure 1.2: The HeaveLock valve in the open position. The cross sectional flow area can be changed by rotating the covering disc.

Figure 1.3: The HeaveLock valve as seen from the side. The bottom section is where the motor and gear will be connected to the valve.

munication possibilities with downhole equipment is limited. With that in mind, the new design should limit the communication to a mere start/stop message, which might include some initial conditions for the operation at hand (e.g. mud flow rate). This implies that motion prediction calculations, valve controllers and dynamic parameter calculations has to be implemented into the logic of the HeaveLock.

1.3 Problem Formulation and Objectives

The objective of this thesis is to implement a control system on a microcontroller board, and minimize the communication between the PC/Simulink [30] (which mimics the topside of a drilling rig) and the microcontroller board (which mimics the downhole of an oil well), thereby making the HeaveLock choke valve autonomous. The objectives that are subject to this thesis was handed out by Professor Ole Morten Aamo, and are listed below:

1. Review background, set the stage for the work to be done (describe lab-setup and its current configuration).
2. Consider the interface/protocol between topside and downhole. Decide which functions that should be implemented on the microcontroller, and which functions that should stay in the topside control system.
3. Change the valve actuator to a design which is able to operate the valve under full pressure.
4. Consider sensor input requirements to the board, and describe alterations needed to the lab.
5. Make hardware decisions (type of board, accelerometer).
6. Implementation (choice of algorithms should be discussed).
7. Test every aspect of the software on the board (dry test in office).
8. Install the microcontroller in the lab (help provided from IPT for this), and test the performance of the control system in the lab.
9. Write a report.

1.4 Guidance for the Reader

The listing below is a guidance for the reader:

- Chapter 2 presents the velocity estimation equations that are used in the project, and how they have been discretized.
- Chapter 3 gives an overview of the entire system, and describes the main functionality.
- Chapter 4 provides a description of hardware selection and the hardware used in this project.
- Chapter 5 describes the HeaveLock software and presents an overview of the incorporated solutions.
- Chapter 6 presents the computer program developed as a graphical user interface for the HeaveLock.
- Chapter 7 shows how the system should be installed, how it can be tuned, and which parameters that can be set.
- Chapter 8 describes the tests that were run in order to confirm correct behaviour of the HeaveLock.
- Chapter 9 provides a summary and conclusion of this thesis in addition to proposed future work.

Chapter 2

Estimation of BHA Velocity

To be able to compensate for heave induced pressure fluctuations, the velocity of the BHA movement has to be known. The velocity is not possible to measure directly, instead an accelerometer is used to estimate the current velocity. A good velocity estimate is critical to be able to reduce the pressure fluctuations as much as possible.

In general it is impractical to estimate velocity from acceleration. Even if the initial conditions of the system are known, measurement noise will lead to unacceptable drift. The fact that the acceleration measurements originate from wave motion implies that the mean of the acceleration and velocity is zero. By exploiting this extra knowledge about the acceleration input, the drifting of the estimated velocity can be eliminated. This has been done in the development of the filtering transfer function, phase correction, and sea state estimation.

Sections 2.1 - 2.4 presents a summary of the heave filter and phase correction equations deduced by Schaut [42]. The work of Schaut showed promising results in simulations, hence it was decided to use this theory in an implementation and test it in the lab.

2.1 Heave Filter

The heave filter is based on the work by Godhavn [18], and is a combination of a second order high-pass Butterworth filter and an integrator. Equation 2.1 [42, Equation 2.3] shows the transfer function of the filter.

$$H_v(s) = \frac{s}{s^2 + 2\zeta\omega_c s + \omega_c^2} \tag{2.1}$$

where

ζ - Damping coefficient
ω_c - Cutoff frequency

2.2 Phase Correction

Using the heave filter that consists of a high pass filter introduces phase error to the estimated velocity. Richter et al. [41] presents three modifications to a standard heave filter that will reduce the phase error. One of these methods involves using a lead-lag element to compensate for the phase error while not changing the magnitude significantly. Equation 2.2 [42, Equation 2.16] shows the transfer function of the phase correction.

$$G(s) = \frac{a_1 s + a_0}{s + b_0} \tag{2.2}$$

where

$$a_0 = b_0 = \sqrt{2}\frac{\omega^2}{\omega_c} \tag{2.3}$$

$$a_1 = -\frac{\omega^2 + \omega_c^2}{\omega^2} \tag{2.4}$$

Equation 2.3 is [42, Equation 2.19], and Equation 2.4 is [42, Equation 2.20].

2.3 Sea State Estimation

To tune the parameters for the filter and phase correction, information about the current sea state has to be known. This information consists of the dominant wave frequency, ω, and the corresponding amplitude of the dominant wave, A. To find the dominant frequency of the waves, a fast Fourier transform (FFT) is performed on a set of previously recorded acceleration

measurements. The buffered acceleration data is run through a Hamming window, the magnitude of the frequency spectrum is calculated by the use of an FFT algorithm, and then the dominant wave frequency is associated with the largest magnitude in the spectrum.

The amplitude of the dominant wave is then calculated using Equation 2.5 [42, Equation 2.33], where the integrals have been changed to finite sums. This equation will remove any bias from the acceleration measurement.

$$A = \lim_{T\to\infty} \sqrt{\frac{1}{\omega^4}\left(\frac{2}{T}\int_0^T \left(a_{acc} - \frac{1}{T}\int_0^T a_{acc}\,\mathrm{d}\tau\right)^2 \mathrm{d}t - 2\sigma_n^2\right)} \tag{2.5}$$

where

ω	-	Dominant wave frequency
a_{acc}	-	Measured acceleration
σ_n^2	-	Variance of the noise of the accelerometer

If the amplitude of the dominant wave is below a given threshold, A_{min}, the filter and phase correction parameters will remain unchanged. Amplitudes below this threshold are most likely to occur only when the system is at rest, e.g. due to activation of the topside heave compensation system. When the parameters are kept as they were before the activation of this system, the HeaveLock system is ready to estimate velocity as soon as the topside heave compensation system is deactivated.

2.4 Cutoff Frequency

Assuming the damping coefficient is given by $\zeta = \frac{1}{\sqrt{2}}$, i.e. the critical damping coefficient, the only varying parameter in the heave filter is the cutoff frequency, w_c. The optimal cutoff frequency is calculated based on an optimization criterion where the expected error variance of the estimated velocity is minimized. The error between an integrator and the phase corrected heave filter has to be calculated. Due to the phase correction, the error will be zero at the dominant wave frequency, ω. Instead of calculating the error at the dominant wave frequency, the error has to be calculated at two points around it, $\omega_{1,2} = \omega \pm \Delta\omega$. These frequencies are used to approximate the relative improvement, which can be written like Equation 2.6 [42, Equa-

tion 2.24].

$$k \approx \frac{\omega_2|\omega^2 - \omega_1^2| + \omega_1|\omega^2 - \omega_2^2}{2\omega^3} \tag{2.6}$$

Now we have everything necessary to calculate the optimal cutoff frequency, as shown in Equation 2.7 [42, Equation 2.26].

$$\overline{\omega}_c = \sqrt[3]{\frac{S_{nn}}{8\sqrt{2}k^2A^2}} \tag{2.7}$$

where

$\overline{\omega}_c$	-	Optimal cutoff frequency
S_{nn}	-	Spectral density of the accelerometer noise, assumed to be constant
k	-	Relative improvement factor from Equation 2.6
A	-	Amplitude of the dominant wave from Equation 2.5

The cutoff frequency of the filter should be far below the lowest frequency of the wave spectrum, but it must also be limited downwards to avoid a very long settling time. The 2%-settling time of the filter is calculated as shown in Equation 2.8 [42, Equation 2.14].

$$T_{2\%} = -\frac{\sqrt{2}\ln(0.02)}{\omega_c} \approx \frac{5.53}{\omega_c} \tag{2.8}$$

where

$T_{2\%}$	-	2% settling time of the filter
ω_c	-	Cutoff frequency of the filter

To limit the 2%-settling time, a lower bound is added to the calculation of the optimal cutoff frequency, as shown in Equation 2.9 [42, based on Equation 2.15].

$$\overline{\omega}_c = \max\left(\sqrt[3]{\frac{S_{nn}}{8\sqrt{2}k^2A^2}}, \frac{5.53}{T_{2\%,\max}}\right) \tag{2.9}$$

where

$T_{2\%,\max}$	-	Maximum acceptable settling time of the filter

2.5 Discretization of Equations

In a computer system it is only possible to solve discrete equations. In order to implement the transfer functions of the heave filter and the phase correction in the microcontroller software, they had to be discretized. This was done by a phase variable state space realization of the transfer functions, and by using the forward Euler method to approximate the derivative of the states.

A strictly proper rational transfer function can be realized as a phase variable state space as seen in Equations 2.11 and 2.12. If the transfer function is not strictly proper, it must be decomposed using partial fraction decomposition to transform it to a strictly proper fraction added to a constant, as seen in Equation 2.10.

$$G(s) = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_{n-1} s + b_n}{s^n + a_2 s^{n-1} + \cdots + a_{n-1} s^2 + a_n s + a_{n+1}} + d \tag{2.10}$$

$$\begin{aligned} \dot{\boldsymbol{x}}(\boldsymbol{t}) &= \boldsymbol{A}\boldsymbol{x}(t) + \boldsymbol{B}u(t) \\ y(t) &= \boldsymbol{C}\boldsymbol{x}(t) + Du(t) \end{aligned} \tag{2.11}$$

where

$$\boldsymbol{A} = \begin{bmatrix} 0 & 1 & \cdots & 0 & 0 \\ \vdots & \vdots & \ddots & \vdots & \vdots \\ 0 & 0 & \cdots & 1 & 0 \\ 0 & 0 & \cdots & 0 & 1 \\ -a_{n+1} & -a_n & \cdots & -a_3 & -a_2 \end{bmatrix} \qquad \boldsymbol{B} = \begin{bmatrix} 0 \\ \vdots \\ 0 \\ 0 \\ 1 \end{bmatrix}$$

$$\boldsymbol{C} = \begin{bmatrix} b_n & b_{n-1} & \cdots & b_2 & b_1 \end{bmatrix} \qquad D = d \tag{2.12}$$

2.5.1 Heave Filter Discretization

The forward Euler method, with T as the discrete time step, can be written as:

$$\dot{\boldsymbol{x}}(t) \approx \frac{\boldsymbol{x}(t+T) - \boldsymbol{x}(t)}{T} \tag{2.13}$$

And so the state space equation representing the transfer function of the heave filter in Equation 2.1 can be written as:

$$\frac{\boldsymbol{x}(t+T)-\boldsymbol{x}(t)}{T} = \boldsymbol{A}\boldsymbol{x}(t)+\boldsymbol{B}u(t) \tag{2.14}$$

which leads to:

$$\begin{gathered}\boldsymbol{x}(t+T) = \boldsymbol{x}(t)+\boldsymbol{A}T\boldsymbol{x}(t)+\boldsymbol{B}Tu(t)\\ \Downarrow \\ \boldsymbol{x}(t+T) = (\boldsymbol{I}+\boldsymbol{A}T)\boldsymbol{x}(t)+\boldsymbol{B}Tu(t)\end{gathered} \tag{2.15}$$

Since the input only changes at discrete time instants, kT for $k = 0, 1, \ldots$, and the response is computed at $t = kT$, Equation 2.11 can be represented on the discrete form:

$$\begin{aligned}y(kT) &= \boldsymbol{C_d}\boldsymbol{x}(kT)+\boldsymbol{D_d}u(kT)\\ \boldsymbol{x}((k+1)T) &= \boldsymbol{A_d}\boldsymbol{x}(kT)+\boldsymbol{B_d}u(kT)\end{aligned} \tag{2.16}$$

The discrete $\boldsymbol{A}$ matrix becomes:

$$\boldsymbol{A_d} = \boldsymbol{I}+\boldsymbol{A}T = \begin{bmatrix}1 & 0\\ 0 & 1\end{bmatrix}+\begin{bmatrix}0 & 1\\ -\omega_c^2 & -2\zeta\omega_c\end{bmatrix}\cdot T = \begin{bmatrix}1 & T\\ -\omega_c^2 T & 1-2\zeta\omega_c T\end{bmatrix} \tag{2.17}$$

and the discrete $\boldsymbol{B}$ matrix becomes:

$$\boldsymbol{B_d} = \boldsymbol{B}T = \begin{bmatrix}0\\ 1\end{bmatrix}\cdot T = \begin{bmatrix}0\\ T\end{bmatrix} \tag{2.18}$$

While the discrete $\boldsymbol{C}$ and D matrices are set:

$$\begin{aligned}\boldsymbol{C_d} &= \boldsymbol{C} = \begin{bmatrix}0 & 1\end{bmatrix}\\ D_d &= D = 0\end{aligned} \tag{2.19}$$

Inserting Equations 2.17-2.19 into Equation 2.16:

$$y(kT) = \begin{bmatrix}0 & 1\end{bmatrix}\cdot\boldsymbol{x}(kT)+0\cdot u(kT) \tag{2.20}$$

$$\boldsymbol{x}((k+1)T) = \begin{bmatrix} 1 & T \\ -\omega_c^2 T & 1-2\zeta\omega_c T \end{bmatrix} \boldsymbol{x}(kT) + \begin{bmatrix} 0 \\ T \end{bmatrix} u(kT) \tag{2.21}$$

Implemented into a computer system, the state space realization for the heave filter can be written like Equation 2.22:

$$\begin{aligned} y(kT) &= x_2(kT) \\ x_1((k+1)T) &= x_1(kT) + Tx_2(kT) \\ x_2((k+1)T) &= -\omega_c^2 T x_1(kT) + (1-2\zeta\omega_c T)x_2(kT) + Tu(kT) \end{aligned} \tag{2.22}$$

where

ω_c	-	Cutoff frequency	u	-	Accelerometer data
ζ	-	Damping factor	$x_{1,2}$	-	Velocity estimation states
k	-	Incremental time step	T	-	Sample time for the filter
y	-	Estimated velocity output			

2.5.2 Phase Correction Discretization

By doing a partial fraction decomposition on Equation 2.2, we can write $G(s)$ as a strictly proper fraction added to a constant:

$$G(s) = \frac{a_1(s+b_0) + a_0 - a_1 b_0}{s+b_0} = \frac{a_0 - a_1 b_0}{s+b_0} + a_1 \tag{2.23}$$

And by a direct comparison to Equations 2.10 and 2.12 we find that:

$$\begin{aligned} A &= -b_0 & B &= 1 \\ C &= a_0 - a_1 b_0 & D &= a_1 \end{aligned} \tag{2.24}$$

The discrete matrices becomes:

$$\begin{aligned} A_d &= I + AT = 1 - b_0 T & B_d &= BT = T \\ C_d &= C = a_0 - a_1 b_0 & D_d &= D = a_1 \end{aligned} \tag{2.25}$$

Implemented into a computer system, the state space realization for the phase correction can be written like Equation 2.26:

$$\begin{aligned} y(kT) &= (a_0 - a_1 b_0)x + a_1 u \\ x((k+1)T) &= (1 - b_0 T)x + Tu \end{aligned} \tag{2.26}$$

where

u	-	Estimated velocity from heave filter	x	-	Phase correction state
k	-	Incremental time step	T	-	Sample time
y	-	Phase corrected estimated velocity output			

Chapter 3

Interface Requirements

The HeaveLock consists of several separate entities. This chapter will describe each of these entities and how they relate to each other.

3.1 Interface Between Topside and Downhole

The goal of the HeaveLock project is to develop an autonomous system that can detect wave motion and control the flow of drilling mud to compensate for the pressure fluctuations that would have occurred without a compensation system. Communication between the downhole unit and a topside control system is not straightforward. The most common way to communicate with downhole equipment is by using a method called mud pulse telemetry. This method uses pressure variations in the drilling mud to encode data between topside and downhole equipment. A great restriction in this technology is the data rate that is possible to achieve. Conventional equipment can achieve around 3 bits per second, while some recently developed technology can achieve around 20 bits per second [26].

Data rates that are this low require the HeaveLock unit to limit the communication between topside and downhole to a minimum. In the development of the control system that is going to be used at the IPT-Heave Lab, this has been taken into consideration. Some communication has been allowed to configure and tune the HeaveLock unit, and to log data to check the performance of the system. Control of start-up and shutdown sequences are also intended to be placed in a topside system. A complete overview of the system can be seen in Figure 3.1.

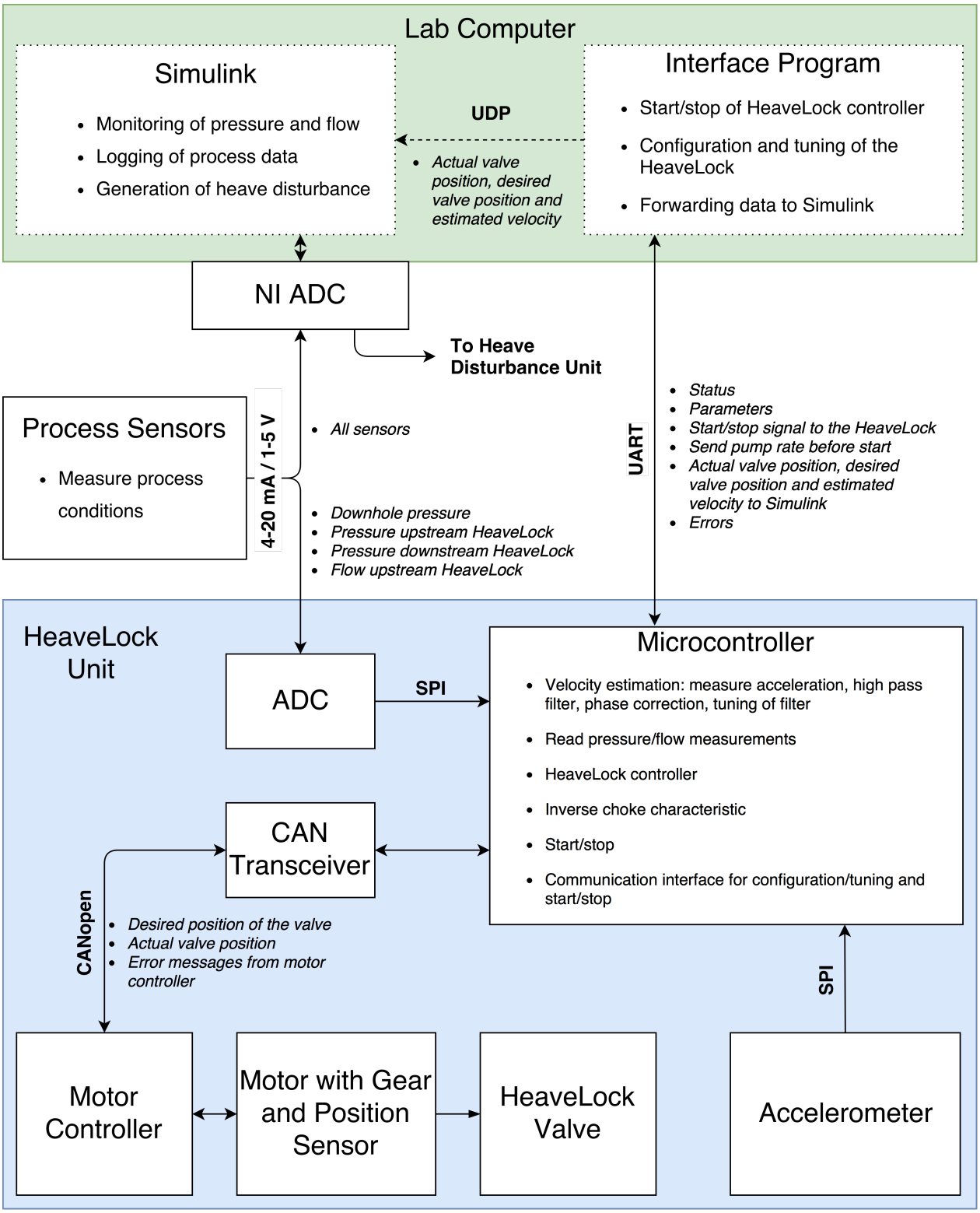

Figure 3.1: Overview of the complete system.

3.2 Simulink

Unlike previous systems at the IPT-Heave Lab, Simulink will not be used to control the Heave-Lock valve. Instead, Simulink is used to monitor and log data from the sensors in the lab and data from the HeaveLock unit. It is also used to control the heave disturbance unit, as disturbance data sets easily can be loaded in or created internally, and sent to an analog output.

3.3 Interface Program

Even though the HeaveLock is designed as an autonomous unit, some monitoring and control of the unit may be preferable. In order to monitor the HeaveLock, a graphical user interface (GUI) is included in the system. This GUI program has been named HeaveLock Control (HLC), and from here on out any references made to this program will use this abbreviation.

HLC's main purpose is to make it easy for a user/operator to get an overview of what the HeaveLock is doing, display any errors that may occur, and present a way of maintaining the HeaveLock, to some extent, without having to change the source code of the HeaveLock.

HLC includes features such as:

- Manually open and close the valve
- Start and stop heave compensation
- Forward live data to the IPT-Heave Lab control system
- Forward valve positioning reference points from the IPT-Heave Lab control system to the HeaveLock
- Display alarms and warnings
- Do a homing-run of the servo to adjust the position interpreted as 100 % open
- Manually set the reference signal with the goal of being able to force the HeaveLock to hold a position, or to be used in order to determine a step response from the IPT-Heave Lab

- Display the filter and phase correction parameters mentioned in Chapter 2
- Set initial filter and phase correction parameters
- Display info-messages produced by the HeaveLock
- Display live data representing the position of the HeaveLock valve
- Display live data representing the setpoint of the HeaveLock valve
- Display live data representing the estimated vertical velocity calculated by the heave filter

3.4 Microcontroller

The microcontroller is responsible for reading acceleration measurements from an accelerometer, estimate the BHA velocity based on the these measurements, and calculate new filter and phase correction parameters. The estimated velocity is used as an input to the HeaveLock controller, where other inputs such as flow and pressure can be used to calculate the desired drill bit flow. These process variables are read using an analog to digital converter (ADC).

The desired valve position is then calculated, by use of an inverse choke characteristic with the desired drill bit flow as input. If the desired valve position has changed, the new position value is sent to the motor controller. In this implementation of the HeaveLock system, the microcontroller also includes a communication interface to be able to control the motor controller, start and stop the heave compensation, and send estimated velocity and valve position to the PC for logging or data processing. Most of this information exchange is done to monitor the performance of the unit, and can be reduced if not omitted at a later stage.

3.5 Motor Controller

The motor controller is used to position the valve by controlling a motor connected to the valve via a gear box. When the microcontroller has calculated the desired valve position, it is sent to the motor controller, where it is used as the reference for the position controller. Position feedback is achieved using a position sensor mounted on the motor. Using an external motor

controller helps to reduce the computation load on the microcontroller, and by choosing a motor controller made by the motor supplier it is possible to get a well tested tailor made solution with a great deal of functionality.

Chapter 4

Hardware

This chapter will elaborate on the choice of hardware and communication protocols that make up the HeaveLock unit. The communication protocols for the peripheral devices are mentioned in this chapter because they can influence the choice of one type of hardware over the other.

4.1 Accelerometer

An accelerometer is an instrument for measuring the acceleration of a moving or vibrating body. Several variants exists, but in order to choose the correct one for an application some information about accelerometers and the application it will be used for has to be known. This section will provide a description of how information about the application was found, and how an accelerometer was selected for the HeaveLock.

4.1.1 Accelerometer Properties

Selecting the correct accelerometer for a certain task is not always straightforward. There are several properties related to accelerometers that needs to be taken into account. This subsection will try to highlight some of these properties.

Measuring Principle

Several principles of measuring acceleration exists, where three common technologies are piezoelectric, piezoresistive and variable capacitance [16]. Piezoelectric accelerometers are the most widely used accelerometers for test and measurement applications. They offer a very wide measurement frequency range (a few Hz to 30 kHz). This makes the principle suitable for both shock and vibration measurements.

Piezoresistive accelerometers are often used to measure shocks, due to their low sensitivity. A typical area of application is in transportation crash tests. The low sensitivity makes the piezoresistive principle less suited to measure vibration. The frequency range is generally wide and goes down to zero Hz. As a consequence these accelerometers are often called direct current (DC) responding.

Variable capacitance accelerometer also have the ability to measure acceleration with frequencies down to zero Hz. They have a high sensitivity and a narrow frequency band. Their susceptibility to temperature changes are also outstanding compared to accelerometers based on piezoresistive and piezoelectric principles. This makes variable capacitance accelerometers suited for measuring low frequency vibration, motion and steady state acceleration.

Number of Axes

The HeaveLock system is designed to detect acceleration in the longitudinal direction of the modeled well in the lab, so the accelerometer does not actually have to have more than one axis. However, using only one axis limits the possible mounting orientation of the unit, and possibly compensation for gravity. If a two or three axis accelerometer is within the same price range, the extra flexibility could be used to remedy the mounting complications.

Interface

Generally speaking, there are two main choices for the interface with an accelerometer, analog voltage or a digital serial protocol. Accelerometers with an analog voltage interface would require a precise ADC, and possibly also an amplifier circuit. Sensors using analog output does not usually process the acceleration signal, requiring this to be done by the circuit built by the

user of the sensor. With a digital serial protocol such as Serial Peripheral Interface (SPI) or Inter-Integrated Circuit (I^2C), all signal conditioning is done by the sensor unit, and the acceleration signal can be read directly using the serial protocol. These units often include extra features such as temperature compensation, alignment calibration, supply voltage compensation and filtering. Since all analog signal processing takes place inside the unit, sensors with a digital serial interface are often less susceptible to noise than their fully analog counterparts. Choosing a digital interface also makes it easier to get complete and accurate noise data from the manufacturer of the sensor, as there will not be any degradation of the signal outside the sensor unit.

Physical Design

Accelerometers are available in several different shapes and sizes. With the development of microelectromechanical system (MEMS), new possibilities of how to produce accelerometers arrived. Many consumer products such as gaming consoles and cell phones are now using accelerometers and complete IMUs, which in turn lowers the unit cost. These MEMS based accelerometers are usually sold as an integrated circuit, which requires the user of the accelerometer to design a circuit board to mount the circuit on. When the sensor is mounted on a circuit board, which in turn is mounted in an enclosure attached to the unit under test, special care has to be taken to ensure the alignment of the accelerometer axes.

Some accelerometers are encapsulated in a metal housing which can be mounted directly on the unit under test. Often these units are factory tested, and axis alignment is within specified limits. The extra housing will also protect the unit from the outer environment, and possibly make the sensor more resilient to electromagnetic interference.

Range

Knowing the max nominal amount of acceleration provides the user of a certainty of resolution. E.g. say an accelerometer with a digital interface is fit for ranges of ±2, ±4, ±8 and ±16 g. This just sets the range of operation, but the number of bits used to transfer the sensor measurements remains the same regardless of range, and so the resolution worsens progressively as the range of operation is adjusted higher. So, knowing the max acceleration that the system will be subjected to can be useful in order to maximize the resolution of the accelerometer.

A crude calculation was done, where the goal was to reveal the maximum acceleration the system would experience under normal conditions. The calculation was done with a simple sinusoidal function as a model of heave motion induced by waves, as seen in Equation 4.1.

$$p = A \cdot sin\left(\frac{2 \cdot \pi}{T} \cdot t\right) \tag{4.1}$$

where

A - Amplitude of movement (travel$_{max}$/2)
t - Time
T - Period of sinusoidal function

The amplitude, A, was set to 40 cm, as the max travel distance in the well in the IPT-Heave Lab is 80 cm. As Albert [1, p.12] mentions in his master's thesis, the shortest period of a waveform to be used in the lab is 3 s. This means that this will produce the highest velocity changes in the system.

To find the maximum acceleration the system will be subject to we can set the jerk $\frac{d^3p}{dt^3} = 0$, and with some simplification and by adding the gravitational acceleration we end up with:

$$a_{max} = \frac{A \cdot 4 \cdot \pi^2}{T^2} + g \tag{4.2}$$

By substituting for the values known for the IPT-Heave Lab we get

$$a_{max} = \frac{0.4 \text{ m} \cdot 4 \cdot \pi^2}{(3 \text{ s})^2} + 9.81 \text{ m/s}^2 \approx 11.56 \text{ m/s}^2 \approx 1.18 \text{ g} \tag{4.3}$$

Note that this is the theoretical acceleration. In a real system there will be some vibration and jerks in the movement, leading to higher acceleration. Estimating how much acceleration that is to be expected from jerks in the disturbance system in the lab is not straightforward. Some safety margins are recommended to make sure that the accelerometer is not saturated. General recommendations have been investigated, and it was found that in systems where sudden starts and stops could occur, a range of ±5 g or more were recommended [29].

4.1.2 Preliminary Accelerometer Test

To verify that the recommended range of at least ±5 g was reasonable, and to get an idea of noise sensitivity of the estimation algorithm, the acceleration in a system similar to the IPT-Heave Lab was measured. The system was set up using hardware that the university and the authors of this thesis had lying around:

- InvenSense MPU-6050 [23] - A cheap inertial measurement unit (IMU) which includes an accelerometer and a gyroscope, mounted on a breakout board. Only the accelerometer was used.
- Arduino Nano [4] - A small prototyping board based on the Atmel ATmega328 microcontroller for collecting and relaying data from the MPU-6050.
- Parker LCB040 [38] with an SMH60 [39] motor - A linear actuator for moving the prototype system up and down with known parameters, used as the wave disturbance. The actuator has a max stroke length of 2000 mm.
- Parker Compax3 I12T11 - Servo drive with position control. The drive has the ability to measure the velocity using an encoder. This velocity is used as the reference for the velocity estimation. The operating instructions for the Compax3 I12T11 can be found online [37].
- Simulink - Used to test the sea state estimation, heave filter and phase correction, described in Sections 2.1-2.4.

To start off, the linear actuator was set to follow a “pseudo-sinusoidal” function decomposed into values for max acceleration, speed and amplitude. Via the serial connection to the Arduino Nano, acceleration data was collected in intervals ranging from 5 to 20 minutes. The data series was then run through the heave filter in Simulink.

As Figure 4.1 shows, the velocity estimation works, with an error within ±0.1 m/s after filter and phase correction parameters have been calculated and the unit has had some time to settle. Smaller deviations in the estimation compared to the reference can be seen, and are likely to be even smaller once a better accelerometer will be used. The deviations may also be a result

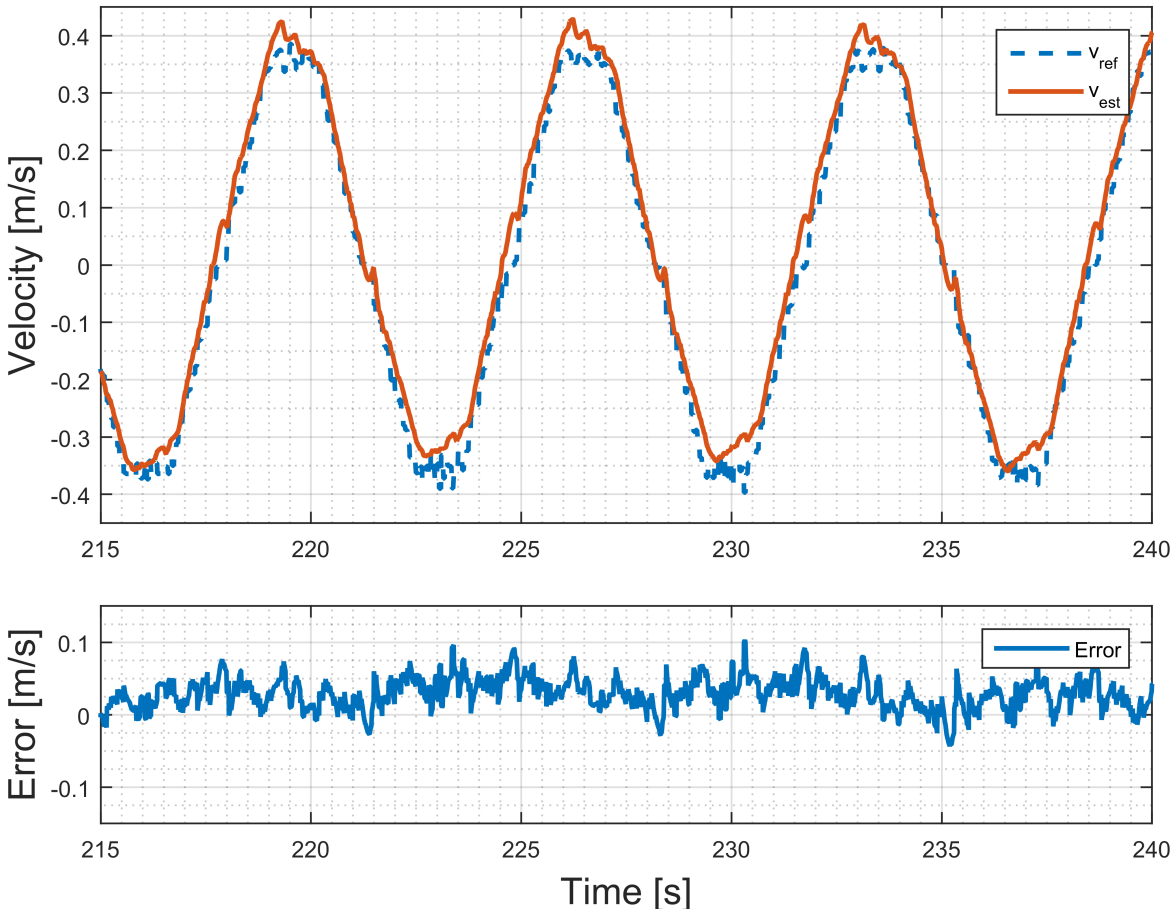

Figure 4.1: Estimated velocity from MPU-6050 accelerometer and reference velocity from Parker Hannifin C3I12T11.

of error in the reference velocity from the linear actuator, as this velocity is the result of a time derivative of a measured position.

Data from the MPU-6050 revealed that real measured acceleration data had peaks of roughly two to three times the calculated maximum acceleration, and made it clear that an accelerometer with a range of approximately ±4 g or more would be necessary to avoid saturation of the sensor.

During testing there was a problem with the IMU chip. The internal processor in the IMU kept freezing, stopping the acceleration measurements at random points in time. The cause of the freezing seemed to be noise from the Parker Compax3 servo drive, but attempts at isolating the noise from the sensor did not yield acceptable results. To overcome this problem, a suitable alternative to the InvenSense accelerometer had to be found.

4.1.3 Choice of Accelerometer

As a reference for what could be expected from the velocity estimation system, details of the Kongsberg Seatex MRU 5+ system was found [27]. This is a motion reference system using MEMS-based sensors to measure attitude and heave motion of marine vessels, developed by leaders in the field for over 20 years. As the unit includes estimation of heave motion it is not relevant as an alternative to an accelerometer, since a part of this project is to test the heave filter and phase correction algorithms. With an acceleration range of ±30 m/s^2 it provides a rough idea of the range required to measure wave motion. One other noticeable characteristic is the very low acceleration noise of 0.0003 m/s^2 root mean square (RMS).

Another accelerometer model that was examined was the Xsens MTi 100, marketed as the "best performing MEMS sensor on the market" [48]. Some key characteristics of this accelerometer is an in-run bias stability of 40 µg and noise density of maximum 150 $\mu g/\sqrt{Hz}$. Communication is possible via several kinds of serial interfaces, using a protocol that requires a software library to be developed on the microcontroller end.

The cost of a Xsens MTi 100 was outside the budget for this project. The institute could lend a unit of this kind to this project, but it would be for a limited period. As the system that is built in the lab is supposed to be used for a longer period, this was not an option.

The accelerometer that is used in this project is the Analog Devices ADIS16448 [2]. It is a complete MEMS based IMU including a triaxial gyroscope, a triaxial accelerometer, a triaxial magnetometer, a temperature sensor, and a pressure sensor. The acceleration range of this unit is ±18 g, which is a bit high, but a relatively low output noise of 5.1 mg RMS and noise density of 0.23 $mg/\sqrt{Hz}$ RMS without any filtering still makes this accelerometer usable for velocity estimation. The sensitivity of the accelerometer is a bit low compared to similar accelerometers with a lower range, but it should be sufficient for a motion detecting application.

Unlike some cheaper units, all sensors in the ADIS16448 have been factory calibrated for sensitivity, bias and alignment, enabling dynamic compensation to temperature changes within -40 °C to +85 °C. To communicate with the unit, SPI is used. Making a software library for a microcontroller to interface an SPI device is quite straightforward, and complete demo libraries for Arduino also exists. The supply voltage required to power the device is 3.3 V, which is common for many microcontrollers on the market.

The unit that was selected for this project had been used in another project at the institute. A benefit of this was that there were no delivery time, and some extra accessories such as a breakout board and cables were already made. As the accelerometer is mounted inside a metal housing, with a ribbon cable connector for the signals, no extra circuit design was necessary. The cost of the unit was a bit high, even though it was a used device. The numerous extra sensors and features are probably causing the high price, which could have been avoided if a sensor only meeting the minimum requirements had been chosen. All in all the positive properties seemed to outweigh the negative, hence this accelerometer was used.

Other MEMS based integrated circuit accelerometers were considered, but due to the bad experience with the MPU-6050's noise tolerance, an accelerometer inside a metal housing was preferred. An integrated circuit accelerometer would also require a circuit board to be designed and manufactured.

4.2 Choice of Microcontroller

To allow for modularity and ease of programming basic functionality into a microcontroller, an Arduino was considered for the task of controlling and operating the HeaveLock. Arduino is a microcontroller system based on hardware and software that is easy to use. All Arduino hardware and software is open-source, making it possible for experienced circuit designers and programmers to contribute to the development of the system. The community that has developed around these prototyping boards, including both beginners and experienced users, is one of the many advantages of Arduino as a system. Most of the trivial problems and challenges have already been solved, and the time required for programming basic functionality is reduced in favor of more time available for solving the main problem at hand.

Using a complete prototyping board, such as one of the Arduino boards, lightens the electronics design workload, and enables more focus on solving the main objectives of this thesis. The way the Arduino boards are built makes it easy to experiment with different electronic circuits, as wires can be connected to the pins of the microcontroller without soldering. Many different expansion boards, called "shields", are also available from the official Arduino store or from other producers. These shields are designed to be stacked on top of the Arduino board,

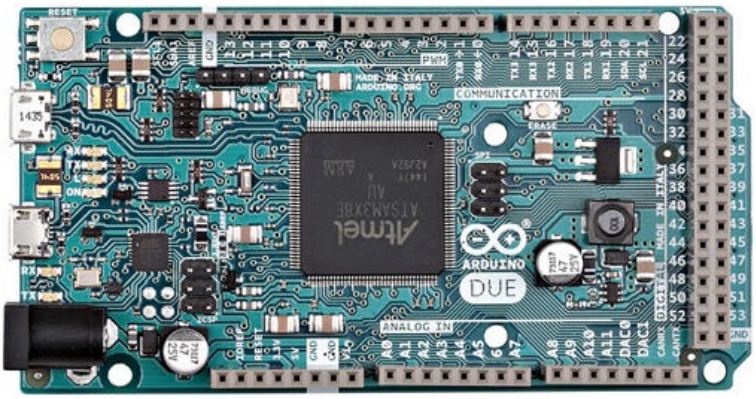

Figure 4.2: The Arduino Due board [6].

making the total solution compact and less fragile than a breadboard circuit. The extra shields may include functions such as WiFi, Ethernet, memory card connection, motor-drivers and relays.

The Arduino boards are pre-loaded with a boot loader, making it possible to reprogram the microcontroller via serial communication instead of using an external programmer. All that is needed to get started is the Arduino board, the free Arduino integrated development environment (IDE) and a Universal Serial Bus (USB) cable to connect the board to a computer. Both communication and power is supplied through the USB cable.

Most of the Arduinos are limited by an 8 MHz or 16 MHz processor, which was assumed to be too slow for the the required tasks in this project. When considering the different Arduino models, the most important properties to evaluate would be speed, memory and the size of the user base. The Arduino Due board, as seen in Figure 4.2, scores quite good on these properties, with an 84 MHz 32-bit ARM Cortex-M3 processor, 512 kB Flash memory and 96 kB static random access memory (SRAM). As the board was launched in the fall of 2012, the user base has had time to develop, and lots of software and hardware have been developed that is compatible with the Due board. The combined properties of the Arduino Due seemed to fit the project requirements, hence it was selected as the microcontroller for the HeaveLock.

4.3 Analog to Digital Converter

In order to control the HeaveLock opening based on a velocity estimate of the HeaveLock movement, a controller to set the opening is required. Depending on the level of complexity of the controller, certain inputs from the lab instrumentation may be needed. The simplest solution is to use a feed forward flow controller and an inverse choke characteristic to determine the valve opening of the HeaveLock. This solution will only require pressure measurements upstream and downstream the HeaveLock. If it is not possible to determine an inverse choke characteristic, a separate flow controller will be necessary to implement. This will require a measurement of the mud flow as a feedback signal to the controller. If the feed forward flow controller does not perform as expected, it might be necessary to change it to a controller with feedback from the downhole pressure.

To allow these possible changes to the control algorithms, the following four process measurements should be available to the microcontroller (see Appendix A for P&ID):

FT2 - Flow upstream HeaveLock

PT6 - Pressure upstream HeaveLock

PT8 - Pressure downhole

PT9 - Pressure downstream HeaveLock

In order to convert the process measurements to digital values, an ADC is needed. The Arduino Due includes an ADC, but the instrumentation in the lab operates at a higher voltage level (10 V) than what the Arduino Due does (3.3 V). A basic voltage divider might give some results, but it is unlikely that these results would be satisfactory. For this reason an external ADC designed for converting analog signals from field instrumentation was ordered. The ADC chosen was the “Extended ADC Shield” from Mayhew Labs [35]. This shield includes the ADC-chip LTC1859, for which a datasheet can be found online [28]. The LTC1859 is a 16-bit ADC, and the inputs can be configured to operate as eight single-ended inputs or four differential inputs. This fits this project, as four inputs is the maximum number of process values required from the lab instrumentation. The LTC1859 can be software-programmed for 0-5 V, 0-10 V, ±5 V or ±10 V input span and operates from a single 5 V supply.

4.4 Motor, Gear and Motor Controller

In the fall semester of 2015 a solution using rotary solenoids to actuate the HeaveLock valve, and a control system controlling the position of the valve was developed by Gundersen [20] and Christensen [14]. When the valve was finally mounted in the lab and tested under pressure, it turned out that it was leaking. To stop the leakage, a new gasket had to be installed, which dramatically increased the torque required to move the valve. The solution using rotary solenoids was not able to deliver the required torque, so a new solution had to be found.

In order to produce sufficient torque for the new valve design, a motor with a gearbox was chosen to control the valve instead of rotary solenoids. Two alternative solutions were presented, one from Harmonic Drive AG [22], and one from maxon motor ag [34]. Both these solutions could deliver sufficient torque, and were within the physical dimensions required by the lab. The crucial factor was the interface, how the motor controller could be controlled from the HeaveLock unit.

The controller from Harmonic Drive AG, HA-680, is equipped with one analog input and also MECHATROLINK CC-Link fieldbus communication. The MAXPOS controller from maxon motor ag is designed to be used with Ethernet for Control Automation Technology (EtherCAT), running the Controller Area Network (CAN) application protocol over EtherCAT (CoE). Using CoE the same communication mechanisms as in CANopen are used, but the physical layer of EtherCAT is used instead of the CAN bus. None of these alternatives seemed to be well suited for the HeaveLock system, as these fieldbus standards are not supported by the Arduino Due hardware. Alternative solutions were investigated, and the EPOS2 motor controller from maxon motors ag was found to fit the project requirements. A picture of the motor controller can be seen in Figure 4.3.

The EPOS2 motor controller supports both RS232 and CANopen interfaces, and is designed to be used as a position controller. Position control is done by a digital proportional, integral, and derivative (PID) controller in cascade with a proportional, integral (PI) current controller. Since the Arduino Due has an integrated CAN controller, only a transceiver chip is needed to be able to use the CANopen protocol. A transceiver is also needed if the RS232 protocol is to be used, as the voltage levels are different from the voltage levels from the Arduino Due.

Figure 4.3: The EPOS2 motor controller used in the project [33].

The DC motor and gearbox came assembled from the factory, and also included an absolute encoder. The encoder is mounted on the motor while the valve will be mounted on the gearbox rotor. The gearbox has a 16:1 reduction, and so the motor will have to turn 1440° in order to turn the rotor of the gearbox 90°, which is the required movement to move the valve from a closed to an open position. This practically renders the absolute encoder properties irrelevant, and so it will have to act as a relative encoder to measure the position of the valve accurately. This means that a homing routine will have to be added to the HeaveLock software and run after every system restart. It will also require that a physical limit has to be mounted on the valve. The homing routine is then able to detect an increase in power consumption when it hits this limit, and thereby know where it is.

4.5 Communication with the Motor Controller

In addition to having a RS232 and CANopen interface, the EPOS2 motor controller can also be controlled by analog signals. A clear advantage of choosing a network/bus communication compared to analog signals is that there is less room for error in the signal. In addition, a network/bus protocol makes it possible to get feedback and configure the device over the same set of wires in real-time. This narrows the choice down to either serial RS232 or CANopen based on CAN.

4.5.1 Comparing RS232 to CAN

Although both RS232 and CAN are serial protocols, they are different from each other in just about every other detail. RS232 can be used with three wires, but also extended to a more complex multi-wire protocol. The standard defines binary 1 as a signal voltage level between -3 V and -25 V, while binary 0 is a represented as a voltage level between +3 V and +25 V. Voltage levels in-between are interpreted as invalid and thereby leaves a margin for noise. Transmitters and receivers are single-ended, which means that noise issues can occur as only one signal line will be affected by noise. The opposite would be differential termination, where both/all signal wires "experience" the same noise, and so the voltage level difference between signal wires remains the same regardless of noise. The fact that RS232 is single-ended makes it sensitive to noise and prohibits the bus from being used with high baud rates and over large distances. RS232 is a single-master/slave protocol, where the master initiates communication and slaves responds to the master. The RS232 protocol can be either full or half duplex.

The physical layer of CAN consists of three wires, CAN-High, CAN-Low and ground (GND). The value of the signal is resolved by the differential voltage between CAN-High and CAN-Low. In other words, CAN is a differential protocol. When the difference is at a maximum (dominant) it is interpreted as a binary 0, while when it is at a minimum (recessive) it is interpreted as binary 1. This enables the CAN protocol to be noise tolerant and enables communication at higher baud rates (1 Mbit/s) at longer distances in noisier environments than RS232. CAN is a multimaster protocol, where all devices can send and receive messages on the bus. All messages are assigned a priority through the use of a message identifier. If several messages are sent at the same time, the message with the highest priority is the one that is sent on the bus. This is the principle of message arbitration. CAN can only be half duplex.

To sum up, CAN is a powerful multidrop communication protocol, which enables reliable communication at a higher baud rate with a higher noise tolerance than RS232. RS232 is easier to implement and all over a simpler protocol than CAN. However, given that the HeaveLock will most likely be mounted in a somewhat noisy environment, and that it is hard to predict the exact length of wires needed for installation, CAN was chosen as the means of communication for the motor controller on the HeaveLock.

4.5.2 CAN Shield Circuit Board

The Arduino Due includes a CAN controller, but needs an external CAN transceiver in order to interface a CAN bus. The CAN controller includes all the features required to implement the CAN protocol defined by the CAN specification for high speeds (ISO 11898-1) and for low speeds (ISO 11519-2). The CAN controller handles all types of frames (Data, Remote, Error and Overload) and can achieve a bit rate of 1 Mbit/s [10, ch.40].

As the CAN transceiver for the HeaveLock, the SN65HVD230 from Texas Instruments was chosen [44]. This transceiver operates at the required voltage level (3.3 V), and is designed for data rates up to 1 Mbit/s.

A CAN shield circuit board was designed to incorporate the CAN transceiver, and to fit the form factor of the Arduino Due. It is a relatively simple circuit that enables the user to turn on or off line termination and change between high speed mode and slope control mode. It also includes header pins for 3.3 V and ground, as the Arduino Due only has one pin for this. The design and final product can be seen in Figure 4.4, with the schematic in Appendix B.

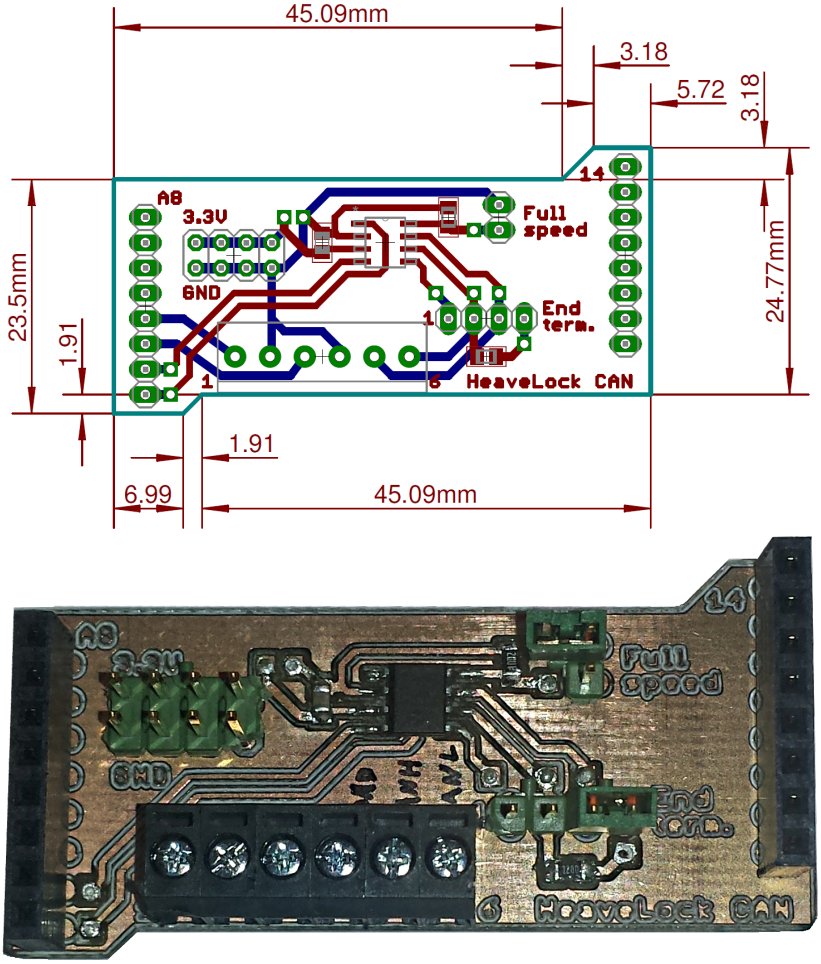

Figure 4.4: The CAN shield board layout (top), and the final product (bottom).

4.6 Communication with the Accelerometer and ADC

The ADIS16448 and the Extended ADC shield communicates over SPI. SPI is a fast synchronous serial data protocol. It was developed for communication between a microcontroller and one or more peripheral devices over short distances [3]. SPI is a full duplex master/slave protocol, which implies that there always has to be one master device which controls the bus. However, several slaves can be controlled by the same master.

The main feature of the SPI-bus is that it is synchronous. This enables the bus to allow for communication without extra overhead (like start/stop bits and agreeing on transmission rate). The fact that SPI is a synchronous bus requires a dedicated line for the clock, in addition to data transmission lines. The clock signal tells each node when to sample the bus.

What makes SPI so popular for small projects and electronics solutions is that the receiver can be a simple shift register. This reduces cost and complexity compared to using the full receiver/transmitter required for other asynchronous bus protocols [43].

The disadvantages of SPI is that it requires more signal lines (master input slave output (MISO), master output slave input (MOSI), serial clock (SCK) and slave select (SS)) compared to other communication methods, like asynchronous serial (recieve (Rx), transmit (Tx)). Also, the communication scheme must be predefined, and the master controls all communication. This prohibits the slaves from communicating with each other.

Chapter 5

HeaveLock Software

This chapter will describe how the HeaveLock software has been developed, how it operates, and how it communicates with other parts of the HeaveLock unit.

The source code that has been written is based on the "self documenting" practice, where there is no exhaustive documentation of every data structure and function. Instead the code should be relatively self explanatory by the use of logical variable names and function interfaces. A list of files that are included in the software can be seen in Appendix C.

5.1 Scheduling and Real-Time Programming

The HeaveLock software will have to do several things, at more or less the same time, in order to gather information, process it, and then act on the processed information. Different tasks and subtasks in the software have different requirements for when to finish, or when to do a new calculation. This implies that something needs to keep track of when tasks and subtasks are running, and when they should run again. In order to do so effectively, a scheduling algorithm should be used to ensure that the different tasks of operations gets the resources (in this case hardware or processing power) required to finish on time.

The by far easiest way to design such a system is by use of a clock. This would involve a variable that keeps track of time, and each time a function is called, check this variable to see if enough time has passed so that the function that has been called is allowed to do its calculations. A far better way is to use a fully fledged scheduler that includes task priorities. Priority

assignment of individual tasks allows for a larger degree of control, and can force the scheduler to assign resources to a task with a higher priority, if two tasks are ready to run at the same time. This feature along with synchronization mechanisms for protection of shared data and resources makes the realization of a real-time concurrent program possible.

There are several scheduling and real-time operating systems (RTOS) out there that fits the purpose mentioned above. FreeRTOS [40] is one of them, and as its name implies, it is free as long as the user do not need certain drivers and complementary software. FreeRTOS has also successfully been ported to the ARM Cortex-M3 core, and can easily be implemented as just an additional library. Based on the arguments above, a decision to use FreeRTOS as a real-time scheduler was made.

5.1.1 FreeRTOS

FreeRTOS [40] is a free real-time micro kernel, made for small to medium sized microcontrollers. The main objective of the kernel is to manage tasks and operations of the microcontroller. FreeRTOS is the free alternative of a family of real-time operating system. OpenRTOS is the commercially licenced variant. OpenRTOS uses the same codebase as FreeRTOS, however, OpenRTOS also includes drivers for Transmission Control Protocol/Internet Protocol (TCP/IP), USB and File Allocation Table (FAT). SafeRTOS is yet another alternative, which is safety-certified according to IEC61508 safety integrity level(SIL) 3.

FreeRTOS is considered to be portable, simple and concise. It offers a smaller and easier real-time processing alternative for applications where heavier operating systems are unavailable, inappropriate, or will not fit. FreeRTOS can be configured to use two different scheduling policies:

- Preemptive - Always runs the highest priority task that is available. Tasks of identical priority share central processing unit (CPU) time (fully preemptive with round robin time slicing).

- Cooperative - Context switches do only occur if a task blocks, or explicitly yields to other tasks.

FreeRTOS is regarded as a mini real-time kernel, rather than a fully fledged operating system, as it lacks much of the functionality often included in an operating system.

Synchronization Mechanisms

FreeRTOS makes available the following synchronization mechanisms:

- Counting semaphores
- Binary semaphores
- Mutexes - differs from binary semaphores as mutexes has priority inheritance
- Message queues
- Task notifications

Ports and Availability

FreeRTOS has been ported to some 20 architectures. The community is large, and it is safe to say that FreeRTOS is a reliable and well tested software solution.

5.2 Task Overview

In order to simplify the design and have better control of timing, the software running on the embedded system has been split into several tasks. The following subsections will closely relate to Figure 5.1, which shows the task division of the HeaveLock software. FreeRTOS has been configured to use preemptive scheduling, and the task numbers in Figure 5.1 (e.g. “T3”) represents the priority assigned to that task, where a higher number represents a higher priority. Arrows that cross the boundary between two tasks are implemented as FreeRTOS queues, with the exception of the arrows intersecting the rectangle at the bottom left of the figure. This rectangle illustrates the separate entities that lies outside of the physical limitations of the microcontroller.

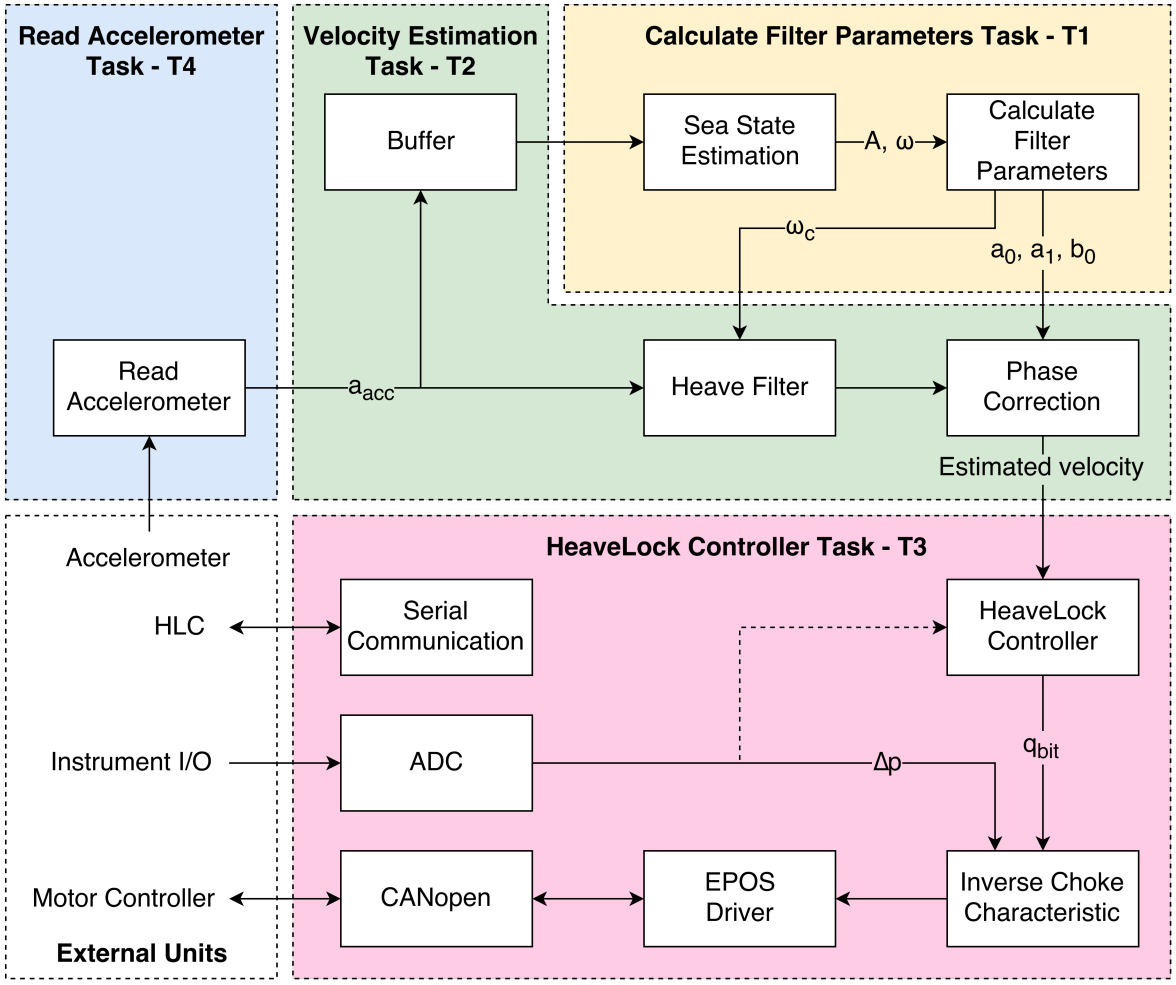

Figure 5.1: Program structure showing the overall data flow and task division.

5.2.1 Read Accelerometer Task

The “readAccTask” (T4 in Figure 5.1) is responsible for reading acceleration data from the accelerometer via the SPI bus when new data is ready. As part of the start-up routine the gravitation is measured, averaging 50 samples over one second, requiring that the unit is at rest during start-up. This gravitational acceleration is then subtracted from every successive measurement.

When the system is running, an interrupt handler is executed after a rising edge on the data ready pin of the accelerometer, and the readAccTask is notified using a task notification. A task notification is used instead of an ordinary binary semaphore, as the task notification is supposed to be 35 % faster and use less random access memory [40]. readAccTask then polls the acceleration value from the accelerometer and forwards it to “velocityEstimationTask” via a FreeRTOS queue.

5.2.2 Velocity Estimation Task

The velocityEstimationTask (T2 in Figure 5.1) reads acceleration data from the readAccTask's FreeRTOS queue every 10 ms. The acceleration values are sent through the heave filter and phase correction equations, Equations 2.22 and 2.26 respectively, thereby updating the estimated velocity. Every 10th acceleration value gets buffered, and when the buffer is full it is sent to the "calculateFilterParametersTask".

5.2.3 Calculate Filter Parameters Task

A task named calculateFilterParametersTask (T1 in Figure 5.1) receives a buffer of accelerometer data, sampled by velocityEstimationTask. The buffered acceleration data is sent through an FFT calculation routine. The FFT returns the frequency spectrum of the acceleration data, and this is used to extrapolate the dominant wave frequency and wave height (the sea state). The dominant wave height and wave frequency, in turn, is used to calculate new filter- and phase correction parameters. These are sent to the velocityEstimationTask by use of a FreeRTOS queue. calculateFilterParametersTask then waits for the next acceleration data buffer to be filled, and then repeats the operation for the new buffer.

5.2.4 Controller Task

The main objective of the "controllerTask" (T3 in Figure 5.1) is to run the HeaveLock controller. The HeaveLock controller receives the estimated velocity via a FreeRTOS queue from velocityEstimationTask and uses this to calculate the desired flow through the HeaveLock. The inverse choke characteristic then uses the desired flow to calculate an appropriate opening of the valve. The controllerTask is also responsible for acquiring the necessary process values from the IPT-Heave Lab, which in unison with the estimated velocity will enable more advanced HeaveLock controller alternatives to calculate the opening of the valve.

The controllerTask is the task that always has the freshest data available, and so it also has the responsibility of transmitting telemetry data to HLC. In addition, the controllerTask checks for new serial messages sent from HLC, and responds accordingly.

5.3 HeaveLock Controller

The controller used to control the HeaveLock is a simple feed forward controller, where the desired flow is calculated based on the volume displacement of the drill string, and the current pump flow, as seen in Equation 5.1. This equation is based on the deductions performed by Schaut [42, Chapter 4]. Volume displacement is calculated using the difference in area over and under the BHA, multiplied with the estimated velocity of the BHA. The flow rate of the mud pump must be known or measured.

$$\hat{q}_{bit} = (A_l - A_u) \cdot 6 \cdot 10^4 \cdot \hat{v} + q_p \tag{5.1}$$

where

A_l	-	Cross sectional area of BHA minus the area of the lower rod
A_u	-	Cross sectional area of BHA minus the area of the upper rod
$6 \cdot 10^4$	-	Conversion factor from m^3/s to l/min
$\hat{v}$	-	Estimated velocity of the BHA
q_p	-	Current pump rate (constant)

5.4 Inverse Choke Characteristic

An inverse choke characteristic is necessary in order to produce a valve opening based on the desired flow rate through the drill bit, $\hat{q}_{bit}$. The choke characteristic was found based on experiments in the IPT-Heave Lab performed by Anders Rønning Dahlen. The experiment involved opening and closing the valve in steps, while measuring flow through and pressure over the valve. By use of curve fitting, Dahlen was able to find a choke characteristic that was similar to the data collection. This characteristic can be seen in Equations 5.2 and 5.3.

$$aq^2 + bq = (cu^2 + du + e)\Delta p \tag{5.2}$$

where

q - Flow through to the drill string in l/min
Δp - Differential pressure over the HeaveLock in bar
u - Desired valve opening (setpoint)

and where a, b, c and d are gains while e is a bias to account for the fact that there is some internal leakage in the valve, which is common for control valves. The various gains and the bias e were identified to be:

$$\begin{aligned} a &= 0.0715 \\ b &= 0.5958 \\ c &= 0.2954 \\ d &= 0.7370 \\ e &= 0.6236 \end{aligned} \tag{5.3}$$

Solving Equation 5.2 for u yields the inverse choke characteristic shown in Equation 5.4.

$$u = -\frac{d}{2c} \pm \sqrt{\frac{d^2}{4c^2} - \frac{e}{c} + \frac{aq^2 + bq}{\Delta p}} \tag{5.4}$$

By using the assumption that the valve opening, u, must be a positive value ranging from 0-1, and that flow always will be positive, i.e. no return flow back up the drill string, we can simplify Equation 5.4 to be:

$$u = -\frac{d}{2c} + \sqrt{\frac{d^2}{4c^2} - \frac{e}{c} + \frac{aq^2 + bq}{\Delta p}} \tag{5.5}$$

Equation 5.5 along with the parameter values in Equation 5.3 is what is implemented into the HeaveLock software.

5.5 CANopen

As mentioned in Subsection 4.5.1, CANopen was chosen as the means of communication with the motor controller. CANopen is one of several commercial versions of the CAN protocol in industrial networks. It is a network that is based on CAN and the CAN application layer (CAL). In other words CANopen implements the layers above and including the network layer of the

Open Systems Interconnection (OSI) model, as seen in Figure 5.2 [24].

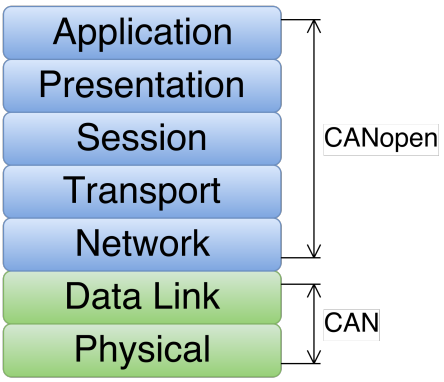

Figure 5.2: OSI model representation of CAN and CANopen.

CAN is a multi-master serial communication protocol that efficiently supports real-time control with a high level of security. The CAN protocol supports four different frame types [10]:

- Data frames - Data frames carry data from a transmitting node to receiving nodes. The maximum data frame length is 108 bits for a standard frame and 128 bits for an extended frame.

- Remote frames - Remote frames are used when a destination node requests data from a source by sending a remote frame with an identifier that matches the identifier of the data frame required. The data source responds by transmitting the requested data frame.

- Error frames - Error frames are generated by any node that detects an error on the bus.

- Overload frames - Overload frames provide extra delay between the preceding and the successive data frames or remote frames.

CANopen was first used for packaging and printing machines, but is now often used in applications such as conveyor belts, automated warehouses, building automation such as heating, ventilation and air conditioning (HVAC), and mobile units such as forklifts and construction machines. By using the CAL, small sensors and actuators may be integrated with PCs at the supervisory control level, into one physical real-time network without requiring gateways [12].

5.5.1 CANopen Device Model

A CANopen device is structured in three main parts, the communication unit, an object dictionary and an application [11]. The communication unit includes a network management (NMT) state machine, communication objects, and the appropriate functionality to transport data items via the underlying network structure.

The object dictionary is a collection of data items, accessible via the network using a 16-bit index and a 8-bit sub-index. It contains configuration data influencing the application objects and communication objects, data describing the current state of the device, and measurement data. When the device is in operational state, interaction between the process and the communication objects and object dictionary are carried out by the application.

The application contains the main functionality of the device, and takes care of interaction with the process environment.

5.5.2 CANopen Communication Objects

CANopen includes several types of communication objects:

- Process data object - PDO
- Multiplex PDO - MPDO - Not included in the HeaveLock software
- Service data object - SDO
- Synchronization object - SYNC
- Time stamp object - TIME - Not included in the HeaveLock software
- Emergency object - EMCY - Not included in the HeaveLock software
- Network management - NMT

To be able to tell apart the different communication objects, all objects are given a communication object identifier (COB-ID). In CANopen the COB-ID is a 32-bit value consisting of some

control bits and a CAN-ID. The CAN-ID is split into a function code and a module-id. The function code identifies the type of communication object used, while the module-id identifies the actual module addressed [31].

To make the motor controller work with the HeaveLock software, only some of the communication objects are needed. Within the included communication objects, only some of the services and protocols have been implemented. The implementation was minimized to reduce the complexity of the software, and to reduce the time it would take to implement a working solution.

Process Data Objects

Process data objects (PDOs) are used to transfer real-time data, with no protocol overhead. Objects in the object dictionary are mapped in corresponding PDOs, to allow the object to be read or written to via PDO protocols. The producer/consumer model can be used to describe PDO communication. Process data can be transmitted from one device (producer) to one or several other devices (consumer(s)), by broadcasting the process data. There are no confirmation of the transmitted data.

Three trigger modes are available to trigger the device to send a PDO message:

- Event- and timer-driven - Message transmission is triggered by an application-specific event or if a specified time has elapsed without occurrence of an event.
- Remotely requested - A message is transmitted as a reply to a remote transmission request initiated by a PDO consumer.
- Synchronously triggered - Message transmission is triggered by a synchronization object.

In the case of the HeaveLock software, the consumer side of the PDO write protocol has been implemented. The PDO transmission is synchronously triggered from the HeaveLock, and the motor controller responds by sending a PDO with up to 8 bytes of application data.

Service Data Objects

A service data object (SDO) is used to provide direct access to entries in the object dictionary of a CANopen device. SDOs may be used to transfer objects of any size, and can be used to configure

the CANopen device. Large objects transferred using SDOs are sent as segments or blocks of data. Prior to sending these large objects, there is an initialization phase where the client and server prepare themselves for the transmission. If the data to be transferred is four bytes or less, it is also possible to transfer the data during the initialization phase. This mechanism is called SDO expedited transfer, and is the mechanism that is used in the HeaveLock unit. Both expedited SDO download and upload has been implemented.

Synchronization Objects

The synchronization object (SYNC) provides the basic network synchronization. One SYNC producer broadcasts the synchronization object periodically, and zero or more SYNC consumers may use this object to initiate their synchronous tasks, such as sending a PDO. To guarantee a short time-delay in synchronous messages, the SYNC is given a very high priority CAN-ID. The SYNC service is unconfirmed.

In the HeaveLock software, SYNC is used to trigger the motor controller to send PDOs containing status word, current position and current operation mode.

Network Management Objects

Network management objects are used to execute NMT services. The NMT is organized with one NMT master and one or several NMT slaves, and through the NMT services the master can change the state of the NMT state machine in the slaves. The state machine and its transitions can bee seen in Figure 5.3 and Table 5.1. The “Operational” state is the only state that allows PDO communication, and when the unit is in the “Stopped” state, only NMT service communication is allowed. Several of the objects in the object dictionary of the EPOS2 motor controller require the state to be “Pre-operational” to allow for changes, such as restoring default parameters and changing mapping of PDOs.

In the case of the HeaveLock software, the motor controller is an NMT slave, and the HeaveLock software is the NMT master. As the master, the HeaveLock software is able to initialize, start, monitor, reset, and stop any node connected to the network using NMT services.

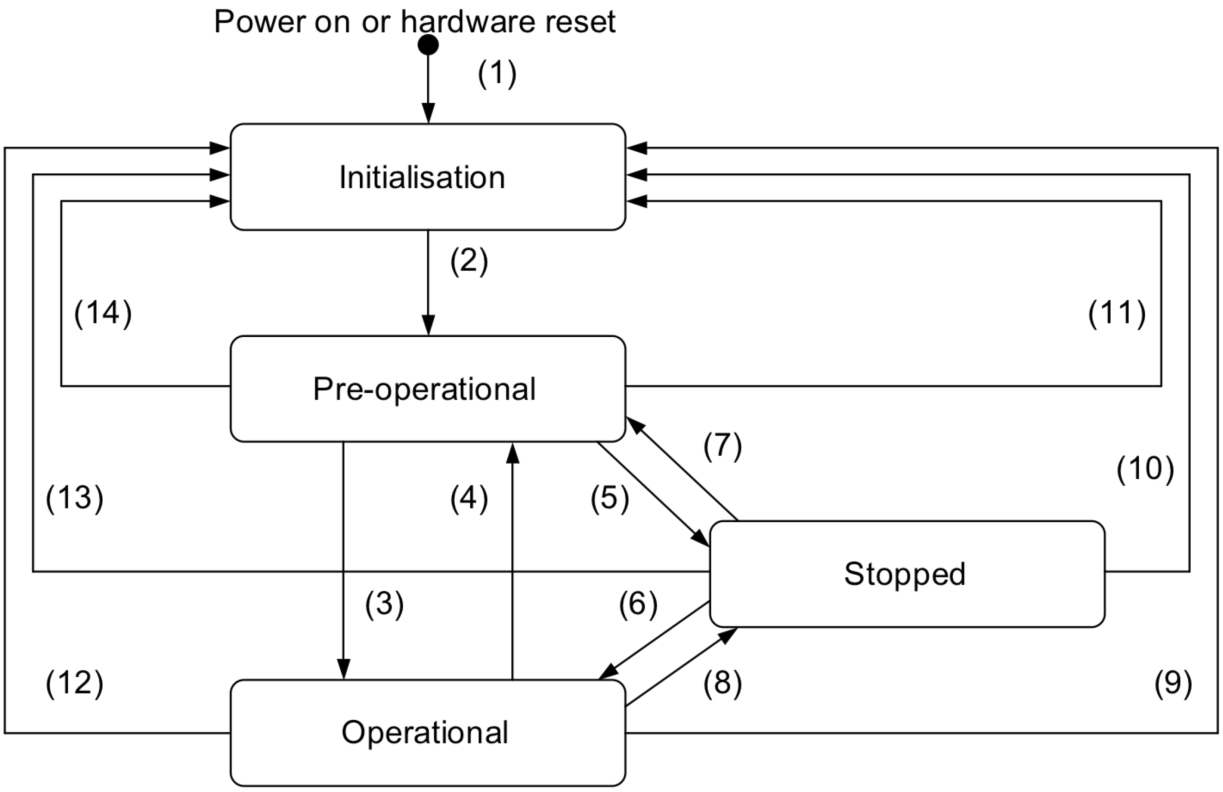

Figure 5.3: NMT state diagram of a CANopen device [11, p.83]. Numbers refer to transitions as described in Table 5.1.

Table 5.1: State transitions of the NMT state diagram seen in Figure 5.3 [11, p.83].

(1)	At Power on the NMT state Initialisation is entered autonomously
(2)	NMT state Initialisation finished - enter NMT state Pre-operational
(3)	NMT service start remote node indication or by local control
(4), (7)	NMT service enter pre-operational indication
(5), (8)	NMT service stop remote node indication
(6)	NMT service start remote node indication
(9), (10), (11)	NMT service reset node indication
(12), (13), (14)	NMT service reset communication indication

5.5.3 EPOS2 Object Dictionary

The motor controller has to be configured before it can be used with the motor, encoder and HeaveLock software. There are at least two ways to do this configuration. Either the wizards in EPOS Studio can be used, or the objects in the object dictionary can be changed manually. To be able to change the configuration, the motor controller must be in NMT state Pre-operational. This is achieved by restarting the motor controller. A summary of objects that have to be changed from their default values can be seen in Appendix D.

5.6 Software Libraries Used in the Project

Not all source code used in this project has been developed from scratch. By using libraries developed by others, more time has been used at a higher level of development, focusing on the main algorithms of the HeaveLock unit. The following subsections will describe the libraries used in this project that have been developed by others, or have been developed based on the work of others.

5.6.1 ADIS16448

The library used to interface the accelerometer has been developed from an example by Chong [13], released under the terms of the GNU Lesser General Public License. To make it work with the Arduino Due running FreeRTOS, and to make it fit the needs of this project, it had to be quite substantially modified. The modified files are included in the source code for this project.

5.6.2 arm_math

The task responsible of calculating filter parameters uses some functions from the Cortex microcontroller software interface standard (CMSIS). Especially the digital signal processing (DSP) library includes useful functions such as FFT and other functions operating on an array of data, all interfaced through the “arm_math.h” header file. Version 1.0.10 of this library is included in the Arduino IDE, and is available after some modifications as described in Appendix E. The included licence permits to use the library to develop software applications for use with micro-

processors manufactured under licence from ARM, such as the Arduino Due that use an ARM Cortex-M3 processor.

This library was chosen for the FFT implementation, as it has been developed to run on processors such as the Cortex-M3 used in this project. Other FFT implementations seemed to be harder to integrate in a project using a microcontroller, especially when the Arduino IDE handles most of the build process.

5.6.3 due_can

The library that interface the integrated CAN controller in the Arduino Due has been developed by Kidder [25], released under the terms of the GNU Lesser General Public Licence. Some research for CAN libraries developed for the Arduino Due led to a forum thread from November 2012 where the development of a library was just started. After some development effort involving several well known Arduino developers, the forum seem to agree to support the use of the library developed by Kidder.

5.6.4 ExtendedADCShield

The library for the ADC shield is inspired by an example library developed by Mayhew Labs [35]. Also this library needed some modification to work with the Arduino Due and FreeRTOS, and some modifications have been done to make the source code more similar to the rest of the source code of the project. The modified library files are included as part of the source code for this project.

5.6.5 FreeRTOS_ARM

The port of FreeRTOS for the Arduino Due has been done by Greiman [19]. It includes version 8.2.3 of FreeRTOS, complete with a standard configuration file. The standard configuration file was replaced with a project specific version to better fit the requirements for this project.

5.7 Development Tools

This section will list the development tools used in this project. Version number of compilers, libraries and integrated development environments (IDEs) may be useful in case backward compatibility has not been handled by the developers of the development tools.

5.7.1 Arduino Software

The Arduino software [5] is required to be able to use the standard Arduino software libraries and the library specific to the Arduino Due board. Version 1.6.7 of the software was used in the code development. The Arduino software includes a very simple IDE, with limited keyword highlighting, and no extra functions. Also no errors will be displayed until the code is built. These are some of the factors that led to a different choice of IDE for the software development.

5.7.2 Visual Micro in Visual Studio

In order to enable a more efficient IDE, the Visual Micro plugin [46] for Microsoft Visual Studio [36] was used to develop the HeaveLock software. Visual Studio incorporates a technology called IntelliSense, which includes useful functions such as word completion and interface information for functions that are used. Syntax errors will instantly be highlighted, and it is easy to navigate in large projects such as the HeaveLock software. The relevant version numbers are listed below:

Microsoft Visual Studio Community 2015 Update 2 with .NET 4.6

Visual Micro (Arduino IDE Extension for Visual Studio) version 1602.7.1

5.7.3 EPOS Studio

EPOS studio is a GUI for configuring and maintaining the EPOS motor controller. This tool is particularly useful for configuring the motor controller with the configuration required for correct operation. The version number used in this project was V. 2.1 Revision 1. EPOS Studio can be downloaded from the maxon motor ag homepage [32].

5.7.4 Git

Git is a version control system. It is free and designed to handle projects of all sizes with speed and efficiency [17]. Git allows for branching of projects making it possible to branch out a project for each feature added to a project, and then merge the complete solution.

The software developed in this project was written by both authors of this thesis. This led to version control being a subject of interest. BitBucket [8], a web-based hosting service for Git projects, was used for version control and as a common workplace.

Since the hardware was moved back and forth between the office and the IPT-Heave Lab different software versions were necessary depending on the hardware available at the time. The branching mechanism of Git made this possible without manually having to synchronize project and source files.

5.7.5 SourceTree

SourceTree [9] is a free Git and Mercurial client for Windows and Mac. SourceTree includes a feature that visualizes the Git-flow, which enable efficient maintenance of Git repositories stored on BitBucket.

Chapter 6

HeaveLock Control

A GUI program named HLC was developed for the HeaveLock. HLC is an event driven application developed in C#, which uses multithreading (with multi-core possibilities) in order to communicate efficiently on the serial bus and minimize the response time of actions made by the user. This chapter will provide an overview of this program and its core functionality. It will also elaborate on the implemented communication solutions developed specifically for this system.

6.1 Basic Functionality and Use

In this section, Figure 6.1 showing HLC in operation will be used as a reference point. A description covering basic functionality and use for each part of HLC will follow. The user interface has been divided into parts depending on rate of use and functionality. The divisions, also known as “group boxes”, all have headers that provide a general description of what functionality lies within a given group box.

Usability has been increased by limiting the options a user has while running HLC. As an example, when the heave compensation has been started it is no longer possible to manually force the valve to an open or closed position, as this would produce unwanted pressure/flow responses in the lab.

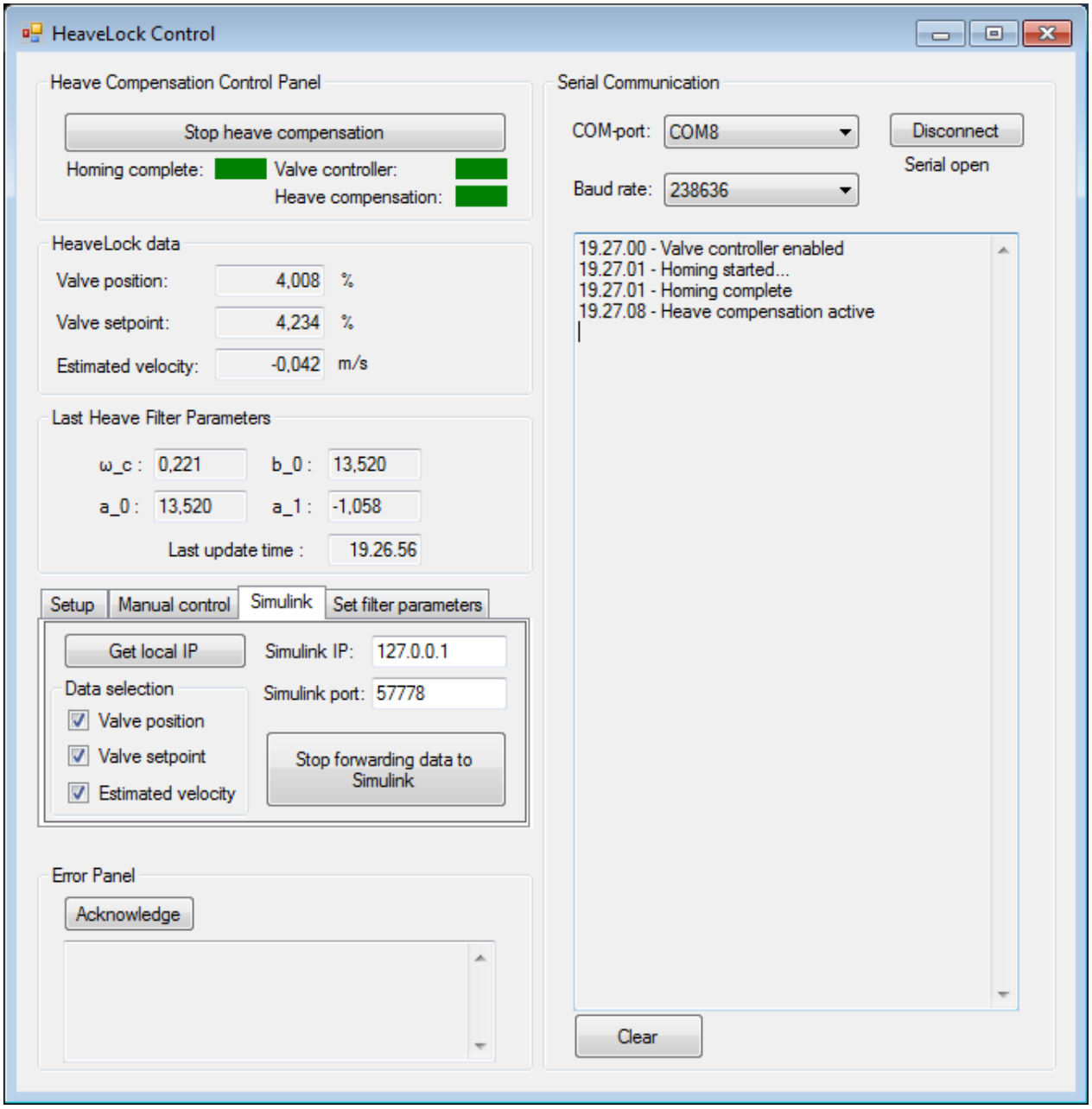

Figure 6.1: Heavelock Control (HLC) in operation.

6.1.1 Serial Communication

On the right side of Figure 6.1 the “Serial Communication” section can be seen. The user will have to choose a COM port and the correct baud rate, and then click the Connect-button. This will initialize the serial communication. When a connection has been established HLC will start polling HeaveLock for a status update every 250 ms until a status has been resolved. An answer from the HeaveLock will result in a color change of the indicators in the “Heave Compensation Control Panel” group box. This usually happens right away, but deviations from this may occur at power-up or if serious faults has arisen.

The large text box on the right will show status and info messages sent from the HeaveLock. It can be cleared by pressing the Clear button at the bottom of Figure 6.1.

6.1.2 Heave Compensation Control Panel

The Heave Compensation Control Panel is meant as the minimum overview needed for a user when the HeaveLock operates autonomously. It includes a button that can start and stop the heave compensation, i.e. put the HeaveLock in auto or manual mode, respectively. The colored indicators (colored green in Figure 6.1) signifies the state of the valve controller (motor controller), heave compensation active/inactive (auto/manual) and if a homing run is required.

For the valve controller and heave compensation indicators: a green light means active, a red light means inactive, while a yellow light will signify that the status is unknown. For the homing indicator a green light means that homing has been done since last restart of the system, while a red one tells the user that a homing run has to be done before any movement of the valve can be allowed. The homing indicator will turn yellow while a homing run is in progress.

Under normal operation these indicators will be yellow until HLC has established a serial connection with HeaveLock. A yellow light on the valve controller while a serial connection is established may indicate a serious fault on the controller. If a complete restart of the HeaveLock and motor controller does not resolve this issue, connecting a PC to the motor controller via USB (EPOS studio) may be necessary in order to acquire information on what is awry.

6.1.3 HeaveLock Data

"HeaveLock data" presents live telemetry data from the HeaveLock unit. The telemetry data contains measured valve position, the reference signal for the valve calculated by the HeaveLock, and the estimated velocity, $\hat{v}$, calculated by the heave filter implemented in the HeaveLock software.

6.1.4 Last Heave Filter Parameters

"Last Heave Filter Parameters" presents the last heave filter and phase correction parameters calculated by the HeaveLock. A datafield shows the time (24-hour clock) of the last parameter arrival. This field was added so the user can know when to expect an update. The parameters are recalculated every 204.8 seconds (or roughly 3 minutes and 25 seconds). This time may seem arbitrary, but it is a direct consequence of sampling time of the accelerometer and the buffer size required for the FFT calculation used in the parameter calculation. For details about the buffer size selection, see Subsection 7.3.3.

Remark: If the HeaveLock is not moving it will not calculate new parameters, and so naturally they will not be updated in HLC.

6.1.5 Setup

"Setup" is one of four panes on the left in Figure 6.1. The Setup pane is shown in Figure 6.2. Setup allows the user the possibility to enable and disable the valve controller (when it is disabled, the valve is in neutral and will move when a force is applied to the rotor). In addition the user can perform a homing run of the valve, which is necessary for each complete restart of the system. Why this is necessary has previously been mentioned in Section 4.4.

The current pump rate, or the pump rate for which an experiment is about to be done, can also be set in Setup. This value has a default value of 15 l/min. This pump rate is the one that is used in the simple HeaveLock feed forward controller, Equation 5.1.

Setup also provides the possibility of altering the max acceleration variable of the motor controller. This value has been set default to 2000 rpm/s. The max acceleration can be set to a

higher value for a quicker valve position response, but unfortunately this will increase vibrations and probably downgrade life-span of the motor, gearbox and valve.

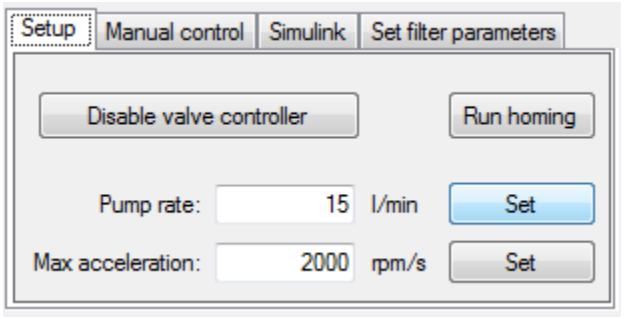

Figure 6.2: The pane in HLC that allows for configuration of motor controller and pump rate.

6.1.6 Manual Control

The "Manual control" pane, shown in Figure 6.3, allows for manual control of the valve position. The valve can be opened and closed by clicking on the "Open valve" and "Close valve" buttons, and a specific position can be set by selecting a setpoint (0.00-100.00 % open) and clicking "Set position".

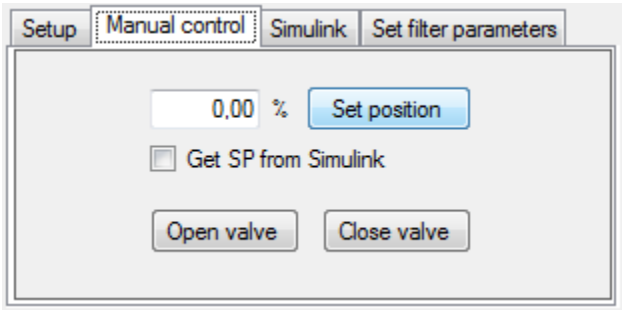

Figure 6.3: The pane in HLC that allows for manual positioning of the valve.

An option was added for sending valve positioning setpoints continuously from Simulink to the HeaveLock through HLC. This way of setting the setpoint has the same basic functionality as setting it manually by clicking "Set position", but the rate of change is determined by the transmission rate of new setpoints in Simulink. This transmission rate can not be higher than 100 setpoints/s, as this would lead to overflow of the serial buffer on the HeaveLock and consequently lead to positioning delays. Although this feature is against the principles of reducing

the communication with the HeaveLock unit, it has been implemented for practical reasons. It is not a feature that is necessary for the HeaveLock to work, and it is disabled during heave compensation.

Sending the valve position setpoint from Simulink was used to test the tracking ability of the valve positioning, as can be seen in Section 8.1. This feature also enables the operator of the IPT-Heave Lab to implement ramping functionality of the valve, which could be used to set the stage for an experiment, and thereby reduce any sudden pressure changes that might occur once heave compensation is activated. A further development of the HeaveLock software could implement this ramping feature downhole.

6.1.7 Simulink

For analytical and logging purposes a feature was added to HLC that enables forwarding of live telemetry data over User Datagram Protocol (UDP). This enables the IPT-Heave Lab control system to store and log telemetry data via a UDP-receive block in Simulink. UDP is regarded as a connectionless protocol. As data is transferred without an established connection, the transfer is more efficient and generally faster compared to Transmission Control Protocol (TCP)[47, p.121]. The trade-off in using UDP compared to TCP in order to gain the higher speed is that there is no guarantee that network packages will not be missed. It was decided that using UDP is reasonable. If a package is lost in the network there is no point in retransmitting it, as the next package will overwrite the last one on the receiving end.

The HeaveLock sends updated telemetry data every loop iteration, this causes an event to be raised in HLC and the event triggers the forwarding to Simulink, thereby minimizing time delay through HLC.

Figure 6.4 shows how a user can forward data over UDP. The IP-address of the PC running Simulink has to be known, either it is at local host or somewhere else on the network. Also, the user has to set the port at which Simulink listens to incoming data. The data forwarded over UDP is a byte array containing 3 floats (datatype single in Simulink). The first four bytes in the byte array are the valve position, the next four bytes represents the valve setpoint, and the last four bytes are the estimated velocity calculated by the heave filter. The user can select which data to send by checking or unchecking the check-boxes on the left of Figure 6.4. If one of

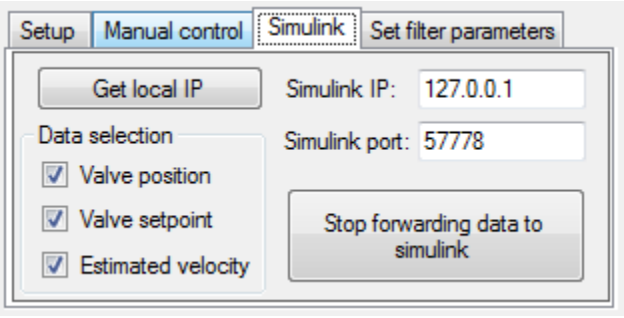

Figure 6.4: The pane in HLC that allows for forwarding data to Simulink.

the check-boxes is unchecked the respective data field in the message to Simulink will contain zeroes. This means that regardless of what the user choose to send, three float values will be sent, and so no alterations are needed in Simulink for handling multiple data structures.

6.1.8 Set Filter Parameters

Figure 6.5 shows the "Set filter parameters" pane. It allows the user to manually set heave filter and phase correction parameters, as long as heave compensation is not active. By setting new parameters the acceleration data buffer is reset, i.e. from the time at which the new parameters are set 204.8 seconds will pass before the HeaveLock recalculates new parameters and thereby overwrite the parameters set by the user.

This option was added so that a user can set a parameter set for a motion profile. As a result the user does not have to wait for the HeaveLock to calculate new parameters each time the HeaveLock has been reset.

The max 2% settling time of the filter, $T_{2\%,\max}$ in Equation 2.9, can be set in this pane. This allows the user to adjust the max settling time of the heave filter in seconds. The default value

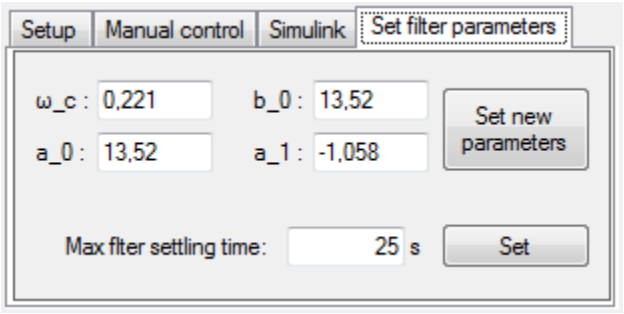

Figure 6.5: The pane in HLC that allows for configuration of filter parameters.

of this parameter is 25 s. More on why this parameter is available in Section 7.3.

6.1.9 Error Panel

At the bottom left of Figure 6.1 the Error Panel can be seen. If a new error message arrives a relatively large bar to the right of the Acknowledge button will become yellow, signaling that there are error messages in the textbox below that has not been acknowledged yet. A wide range of error messages has been pre-programmed into both the HeaveLock software and HLC. A list of these can be seen in Appendix F.

6.2 Operating Procedure

This section will provide a list of actions that needs to be taken to make the HeaveLock operate after a restart or shutdown.

1. Make sure the HeaveLock is connected with a USB cable to the PC. Start HLC.exe.

2. Select the COM port that the HeaveLock is connected to, and the correct baud rate (238 636) from the pull down lists in the top right corner of Figure 6.1.

3. Click the Connect button. All the status indicators in the top left corner of Figure 6.1 should now be red. If any of them are yellow, see Subsection 6.1.2.

4. Select the Setup pane on the left of Figure 6.1. Click the “Enable valve controller” button shown in Figure 6.2. The text on the button should now change to “Disable valve controller”, and the status indicator of the valve controller in the top left of Figure 6.1 will turn green.

5. Do a homing run of the valve by clicking the “Run homing” button in the Setup pane.

6. Wait for the homing routine to complete. The “Homing complete” indicator should switch from red to yellow (while homing), and from yellow to green when homing has been completed.

7. The HeaveLock can now be used for either manual control, see Subsection 6.1.6, or automatic heave compensation, see point 8.

8. Automatic heave compensation

   (a) Select the pump rate to be used for the upcoming experiment. Pump rate can be set in the Setup pane by typing in the pump rate in the correct text box in Figure 6.2, and then click the corresponding "Set" button. This is crucial, as this directly influences the controller used to position the valve, see Equation 5.1.

   (b) Start the automatic heave compensation by clicking the "Start heave compensation" button on the top left of Figure 6.1. When heave compensation has been started, the heave compensation indicator will turn green.

If the user disconnects from the HeaveLock, and the HeaveLock is not turned off or reset, the states of the HeaveLock will persist. When HLC reconnects, it will poll status information from the HeaveLock and update the status indicators. This means that HLC can be used to set up the HeaveLock and then start the heave compensation. HLC can now disconnect and the HeaveLock will operate autonomously. However, in order to acquire live data from the HeaveLock, HLC has to be running and connected to the HeaveLock.

6.3 Serial Protocol Design and Implementation

The communication protocol used between HeaveLock and HLC is a serial communication over USB, and universal asynchronous receiver/transmitter (UART). The bus is set up with 1 start bit, 8 data bits and 1 stop bit. The baud rate has been set to 238 636 bps. A special set of commands and predefined messages has been implemented to enable safe and efficient communication. The next sections will elaborate on the data packets that are sent between HLC and the HeaveLock, with the goal of supplying enough information for any further developers with access to the source code to be able to use the protocol.

6.3.1 Serial Data Protocol

An overlaying protocol for the serial communication has been developed and implemented. Figure 6.6 and Table 6.1 shows the buildup of a serial message being sent between HLC and the HeaveLock. The “command” field specifies what type of message is following, and thereby supplies enough information to inform the receiving end of how to decode the message. The “length of data” field specifies the length of the data in the “data” field, and is there to ensure that all data is received. The “data” field has a limitation of 255 bytes (or 255 characters for string based messages). The length is limited by the fact that the “length of data” field is 1 byte, and therefore can not have a value larger than 255. The “header” and “tail” fields enable messages of different lengths to be sent. When both header and tail has been found, the message is the data that is between the header and tail.

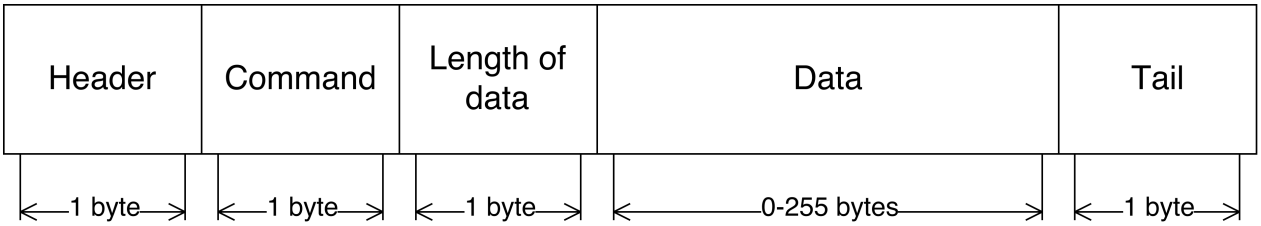

Figure 6.6: Buildup of a serial data packet.

Table 6.1: Detailed buildup of a serial message.

Name	**Size**	**Value**
Header	1 byte	0xFE
Command	1 byte	See Tables 6.2 and 6.3
Length of data	1 byte	0x00-0xFF (0-255)
Data field	0-255 byte	0x00-0xFF (0-255) per byte
Tail	1 byte	0xFD

6.3.2 Serial Commands

This section presents the commands that are implemented into the HeaveLock software and HLC, which enables communication between the two. Table 6.2 shows the commands sent from HLC to the HeaveLock software, while Table 6.3 shows the commands sent from the HeaveLock software to HLC.

Table 6.2: Commands used in messages from HLC to HeaveLock.

Command	Name	Length of data	Data type
0x02	Stop heave compensation	0 byte	N/A
0x03	Start heave compensation	0 byte	N/A
0x06	Start homing of valve motor	0 byte	N/A
0x09	Enable valve controller	0 byte	N/A
0x0A	Disable valve controller	0 byte	N/A
0x11	Open valve	0 byte	N/A
0x12	Close valve	0 byte	N/A
0x14	Set valve set point	4 byte	float
0x15	Get status	0 byte	N/A
0x21	Get position, setpoint and estimated velocity	0 byte	N/A
0x25	Set valve set point (no confirmation)	0 byte	N/A
0x27	Set max 2% settling time for filter	4 byte	float
0x28	Set pump rate	4 byte	float
0x29	Set ω_c, a_0, a_1 and b_0	16 byte	4 x float
0x34	Set max acceleration of motor	4 byte	float
0x77	Acknowledge errors	0 byte	N/A

Remark: Regarding Tables 6.2 and 6.3: If it says “0 byte” in the “Length of data” column, it means that the “Length of data” field in the message needs to have that value. In such a case the data field itself will be ignored on the receiving end, and so it should be omitted from the message.

Table 6.3: Commands used in messages from HeaveLock to HLC.

Command	Name	Length of data	Data type
0x02	Stop heave compensation confirmed	0 byte	N/A
0x03	Start heave compensation confirmed	0 byte	N/A
0x04	Status message	0-255 byte	char[]
0x05	Information message	0-255 byte	char[]
0x06	Homing started	0 byte	N/A
0x07	Homing complete	0 byte	N/A
0x08	Homing not executed	0 byte	N/A
0x09	Valve controller enabled	0 byte	N/A
0x0A	Valve controller disabled	0 byte	N/A
0x10	General error command	0-255 byte	char[]
0x11	Valve opened	0 byte	N/A
0x12	Valve closed	0 byte	N/A
0x14	Valve setpoint changed	0 byte	N/A
0x21	Position, setpoint and estimated velocity	12 byte	3 x float
0x27	Max settling time change confirmed	0 byte	N/A
0x28	Pump rate change confirmed	0 byte	N/A
0x29	Parameters ω_c, a_0, a_1 and b_0	16 byte	4 x float
0x34	Max acceleration change confirmed	0 byte	N/A
0x77	Error acknowledge confirmed	0 byte	N/A

6.3.3 Time Delay of Live Data

As can be seen in Tables 6.2 and 6.3 most messages that can be sent do not carry data, and therefore just contain 4 bytes (header, command, length of data, and tail). However, these short messages are not sent periodically, and require user input in order to be sent. This means that the time it takes to send and receive 4 bytes every now and again are negligible compared to the telemetry data that is sent from HeaveLock each time it completes a loop of the program. The HeaveLock program loops every 10 ms, which means telemetry data is sent every 10 ms, and so we can find a theoretical average transmission time of one telemetry data message (ignoring signal loss). We can also safely ignore all other messages, as these will just be noise in the collection of transmissions. With these assumptions and the fact that we know the baud rate and how the serial communication is set up (1 start bit, 8 data bits, and 1 stop bit) we can do the calculation:

$$N_{bits/byte} = 1 \text{ start bit } + 8 \text{ data bits } + 1 \text{ stop bit } = 10 \text{ bits/byte} \tag{6.1}$$

$$N_{bytes/message} = \text{header} + \text{command} + \text{length} + 12 \text{ byte telemetry data} + \text{tail} = 16 \text{ byte} \quad (6.2)$$

And so the time to send a telemetry data message can be calculated to be:

$$\frac{N_{bytes/message} \cdot N_{bits/byte}}{238\,636 \text{ bps}} \approx 0.67 \text{ ms/message} \quad (6.3)$$

0.67 ms/message is the theoretical time it takes to send each telemetry data message. This means that the live data presented in HLC (see Subsection 6.1.3) will lag behind the HeaveLock by at least 0.67 ms.

Chapter 7

Installation, Tuning and Adjustments

This chapter will describe how the HeaveLock has been set up and prepared for first use, and what needs to be done in order to implement the HeaveLock into to the IPT-Heave Lab.

7.1 Installation of Hardware in the Lab

The accelerometer is what decides the desired mounting orientation for the HeaveLock. The HeaveLock software does not contain any advanced algorithms for calculating the direction of gravity, and as a consequence it is required that the accelerometer is mounted along one of its three axis. The HeaveLock has been configured to operate on the y-axis, and gravity is defined positive in the opposite direction of the y-axis, see Figure 7.1. This means that to avoid a reconfiguration of the source code, the accelerometer has to be mounted with the pressure sensing opening (hole in the xz-plane) in the upward direction. It is, however, possible to reconfigure the orientation if for practical reasons it has to be mounted in another orientation. This can be done in the main file, “HeaveLockSW.ino”, in the source code at line 22 and 23, shown in Listing 7.1.

```
22  #define ACC_AXIS 'y'
23  #define ACC_SIGN -1 // 1 for positive and -1 for negative
```

Listing 7.1: Configure accelerometer orientation.

The desired axis to be used can be selected by changing “ACC_AXIS” to ’x’, ’y’ or ’z’, and what direction is up by modifying “ACC_SIGN”. The sign of ACC_SIGN relates to the sign of the axis visualized in Figure 7.1.

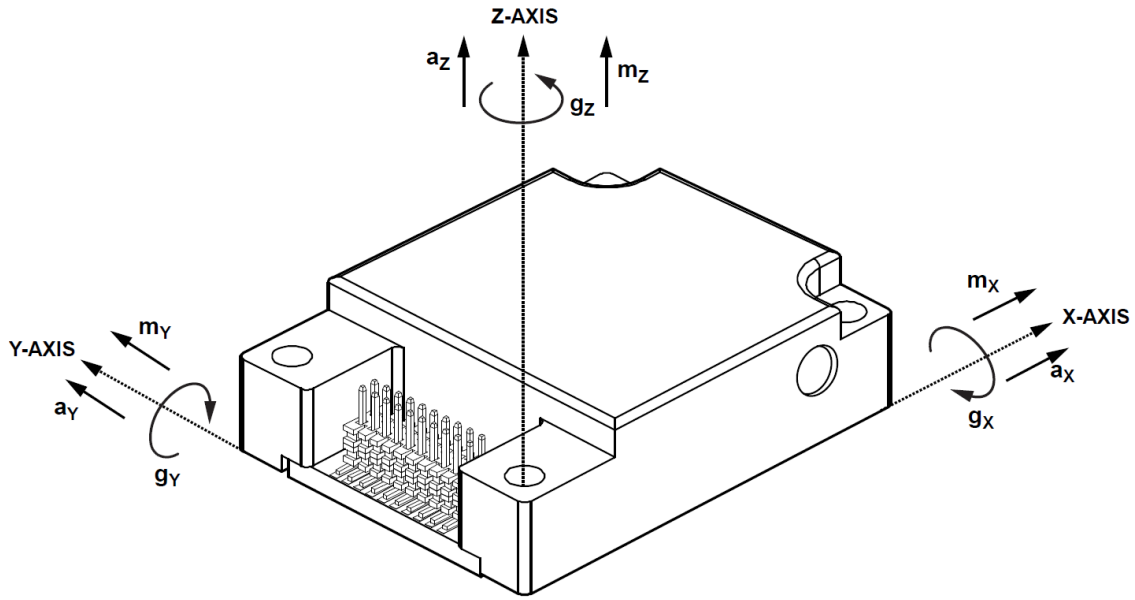

Figure 7.1: Inertial sensor direction reference [2].

7.2 Electrical Connections

Some modification to signal and power wires are required in the lab. Keep in mind that signal cables should be physically separated from all power cables, to avoid interference.

7.2.1 Process Instrumentation

The HeaveLock needs the signal wires from four process instruments:

FT2 - Flow upstream HeaveLock

PT6 - Pressure upstream HeaveLock

PT8 - Pressure downhole

PT9 - Pressure downstream HeaveLock

The instruments can be seen in the P&ID in Appendix A, and it is suggested that a multicore cable (minimum 8 wires) should be pulled from the termination cabinet in the lab to the HeaveLock, containing the signals required from the process instrumentation. The input/output (I/O)-list in Appendix G shows the termination of all the instrumentation in the lab, while Appendix H provides a circuit diagram for the HeaveLock. The circuit diagram shows how FT2, PT6, PT8 and PT9 has to be connected the the ADC shield of the HeaveLock.

7.2.2 Motor Controller

As mentioned previously, the motor controller is controlled via CAN bus. The CAN bus needs to be connected accoring to the circuit diagram in Appendix H.

7.2.3 Power and Communication

In order for the HeaveLock to be able to communicate with HLC, a USB cable needs to be connected to the programming port of the Arduino Due. The programming port is the USB port at the bottom left of Figure 4.2, above the power supply connector.

It is recommended that the Arduino Due is powered by a power supply (7-12 VDC), and not just rely on the power provided by the USB connection. In the lab, where a longer USB cable is necessary, the voltage drop in the USB cable might be severe enough to cause inaccurate measurements by the ADC. By using a separate power supply, and the voltage regulator on the Arduino Due, a more reliable supply voltage can be guaranteed.

7.3 Heave Filter Parameters

This section will highlight the parameters that have been tuned for the lab. These parameters have previously been mentioned in Chapter 2, and what follows will provide a description of why the parameters have been set like they have.

7.3.1 Settling Time

$T_{2\%,\max}$ describes the max settling time allowed for the heave filter. Equation 2.9 uses $T_{2\%,\max}$ to create a lower bound for the cutoff frequency of the heave filter. $T_{2\%,\max}$ has been set default to 25 s. This will set a lower bound for the cutoff frequency at

$$\overline{\omega}_c \approx \frac{5.53}{T_{2\%,\max}} \approx 0.2212 \text{ rad/s} \tag{7.1}$$

This means that all periods longer than ~28.4 s will be filtered away. Setting $T_{2\%,\max}$ to 25 ± 0.5 s yielded the best results under simulation with accelerometer data from the HeaveLock, compared to other values ranging from 5-40 s. This was tested by calculating the RMS error between the estimated velocity calculated by the heave filter and phase correction equations, Equations 2.22 and 2.26, and the measured velocity from the linear actuator. The test was performed with several values for $T_{2\%,\max}$, and for sinusoidal motion periods ranging from 3 s to 10 s. Settling times above 40 s leads to superimposed oscillations in the velocity estimate.

Unfortunately the optimal cutoff frequency calculation, Equation 2.7, does not produce a reasonable cutoff frequency. Instead it generates cutoff frequencies several orders of magnitude lower than what the optimal cutoff frequency should have been, suggesting settling time for the filter in the 250-350 s range. This alters the filter to the extent that it cannot reproduce an estimated velocity. Consequently the lower bound calculation in Equation 2.9 always determines the cutoff frequency. Practically this means that the filter is tuned with a constant cutoff frequency. The ramification of this fact is that the system is sensitive to variations in the input signal. However, the sensitivity is low enough for a constant cutoff frequency to make the heave filter calculate an estimated velocity for periods ranging from 3 s to 10 s.

Since $T_{2\%,\max}$ now determines the cutoff frequency, it has been added as an adjustable parameter in HLC. The operator of HLC can now indirectly adjust the cutoff frequency of the filter, by tuning $T_{2\%,\max}$ based on initial conditions for the experiment to be performed in the IPT-Heave Lab. Figuring out why the optimal cutoff frequency calculation fails and how to fix it is outside the scope of this thesis, but is included as part of the recommendations for further work. With that in mind it was logical to implement $T_{2\%,\max}$ into HLC, rather than an option to change ω_c directly, to keep the intended functionality intact.

The default value of $T_{2\%,\max}$ can be changed by modifying line 18 in "TaskParam.h" in the HeaveLock software source code. Line 18 can be seen in Listing 7.2.

7.3.2 Noise Settings

The variance of the accelerometer noise, σ_n^2, was set to the value found in the accelerometer datasheet, i.e. $(5.1 \cdot 9.81 \cdot 10^{-3})^2$ m^2/s^4 [2]. Likewise the noise power, S_{nn}, was also taken from the datasheet and set to be $(0.23 \cdot 9.81 \cdot 10^{-3})^2$ m^2/s^3 [2]. As can be deduced by Equations 2.5 and 2.7, these equations are dependent on σ_n^2 and S_{nn} respectively, and hence a change in the noise settings will alter the the calculated cutoff frequency and/or the calculated amplitude of the dominant wave. This means that if an accelerometer change becomes relevant in the future, the noise settings must be changed accordingly.

The noise values can be changed by modifying line 19 and 20 in "TaskParam.h" in the HeaveLock software source code, as shown in Listing 7.2.

```
17  #define DW 0.02f * 2.0f * (float)PI   // Frequency range around the
    dominant wave frequency
18  #define T_2MAX 25.0f          // 2 % settling time of the heave filter
19  #define SIGMA2 sq(5.1*9.81*0.001)
20  #define S_NN sq(0.23*9.81*0.001)
```

Listing 7.2: HeaveLock parameters found in "TaskParm.h" in the HeaveLock source code.

7.3.3 Window Length

The window length is the number of accelerometer samples that are stored between calculations of new filter parameters. By changing the window length, the update frequency of the filter parameters will change. The calculation of the amplitude of the dominant wave, A, will also be changed, as the window length defines the length of the finite sums replacing the integrals in Equation 2.5. A value for the window length has to be chosen to make it possible to get a good estimation of the accelerometer bias, allow an adequate part of the current wave frequencies to be sampled, but still small enough to allow frequent update of the filter parameters.

Window length for an FFT is normally chosen as a power of two to increase the algorithm performance. The FFT that is implemented into the HeaveLock allows for data lengths of 128, 512 and 2048 [7], where a length of 2048 was selected based on the previous work of Schaut [42]. With a sampling time for the FFT of 0.1 s it takes 204.8 s to fill the buffer for the FFT. This is believed to provide enough information about the sea state in the lab, as the longest periods will be 10 s. An FFT length of 512 would likewise take 51.2 s to fill up the buffer, and was thought to be to short to provide a decent picture of the oscillations. This, however, could not be tested in the office as the linear actuator used for testing the velocity estimation could only move the HeaveLock with a pseudo sinusoidal positioning. It might be that a window of 512 will suffice, but this has to be tested in the lab.

The window length can be changed by modifying the default buffer length in the HeaveLock software source code. This parameter can be accessed in the “Buffer.h” file at line 12.

7.3.4 Frequency Range for Error Calculation

The frequency range for error calculation was set to $\Delta\omega = 0.02 \cdot 2\pi$ rad/s. This value came from the work by Schaut [42]. Values of 0.03 Hz and 0.01 Hz were tested, but resulted in a larger RMS error between estimated velocity and the reference value, and so 0.02 Hz was used. No further testing on the subject was performed. Changing $\Delta\omega$ will change $\omega_{1,2}$ used in Equation 2.6. This will change the optimal range of the wave frequencies that are expected to occur between calculation of filter parameters.

The frequency range can be changed by modifying line 17 in “TaskParam.h” in the HeaveLock software source code, as shown in Listing 7.2.

7.3.5 Amplitude and Frequency Limitations

The sample frequency, F_s, of the FFT is determined by the sample time of the buffered accelerometer data. The accelerometer is sampled every 0.01 s. Buffering every 10th of these values yields an FFT sample time of 0.1 s, resulting in $F_s = 10$ Hz ≈ 31.416 rad/s. To avoid unnecessary calculations and looping through data sets in the HeaveLock software, an upper limit was set for the frequency. The FFT window length has been set to 2048 data points, where index

2048 represents F_s. As mentioned earlier, the system has been designed to operate on periods of oscillation ranging from 3 s to 10 s. With this in mind an upper limit for the search through the FFT data for the dominant frequency was set at index 2048/8 = 256 in the FFT data array. This corresponds to an upper frequency limit at roughly 7.85 rad/s, or in periods of oscillation: 0.8 s. Well below the minimum 3 s design value.

A lower frequency limit was also set, now starting the search through the FFT data at index 10, which corresponds to a lower frequency limit at roughly 0.31 rad/s. This is roughly equal to periods of 20.5 s, well above the 10 s design value. The main reason for the lower frequency limit is to avoid invalid data. Some low frequency noise is always present, and to stop the algorithm from selecting invalid data as the dominant wave frequency this lower frequency limit is necessary.

When the dominant wave frequency and wave amplitude have been found, the next step is to calculate the filter parameters. To avoid calculation of unreasonable filter parameters, a limit has been set to define the minimum amplitude of a dominant wave, A_{min}. A_{min} has been set to 0.1 m, and ensures that when the motion of the HeaveLock is below 10 cm it will interpret this as being at rest. The HeaveLock will keep its present filter and phase correction parameters, and this way it is ready to compensate for heave motion as soon as the HeaveLock starts moving again.

7.4 HeaveLock Controller

The simple feed forward HeaveLock controller does not contain any other parameters than the cross sectional area of BHA minus the area of the lower rod, A_l, and the cross sectional area of BHA minus the area of the upper rod, A_u. These parameters are found by measuring the diameter of the BHA, the upper rod and the lower rod, and calculating the areas. The file “HLController.h” makes the physical measurements of the lab available.

The software that has been developed is prepared for other controller implementations, such as a different linear, and possibly non linear controller designs. To change the controller type, the new controller has to be implemented as a function, and the function call to “simpleFeedForwardController” must be changed to the new controller implementation. Process pa-

rameters can be accessed using a struct of the type “ProcessMeasurements” called “_process”.

7.5 Position Controller

The position controller running in the motor controller was tuned using auto-tune in EPOS Studio, without the valve attached. The gains of the controllers were determined by the maxon software after running some tests on the motor. If position error turns out to be a problem, the gains of the controllers can be tuned manually using EPOS Studio as an interface to the motor controller.

7.6 Accelerometer

As can be seen in Figure 7.2, the ADIS16448 accelerometer [2] is equipped with two cascaded averaging filters, providing a Bartlett window with a finite impulse response (FIR) filter response. In addition there is a decimation filter on the output, reducing the update rate by averaging the samples. The internal sampling clock is used, sampling the sensors 819.2 times per second.

By collecting data from the accelerometer while it was subjected to heave motion, it was found that some filtering was needed to be able to distinguish the dominant wave frequency from the noise. The amount of filtering is set by programming the number of taps in each stage, $N_B = 2^B$, where the exponent, B, can be specified. The lowest value of the filter size variable B that yielded usable results was 5. By filtering the signal, a phase delay of N_B samples will be added to the signal. Using the internal sampling clock and filter size variable $B = 5$, the phase delay will be

$$2^5 \text{ samples} \cdot \frac{1}{819.2 \text{ samples/s}} = 39.06 \text{ ms}$$

Since the velocity estimation algorithm has a sample time of 0.01 s, there is no need to get a new acceleration reading 819.2 times per second. By reducing the sample time the measured value will be a result of averaged internal samples, reducing the noise of the measurements. As a minimum a new acceleration reading should be available 100 times per second, but this does not add up using the internal sampling clock. The decimation rate $N_D = 2^D$ is set by setting the exponent, D, in a register of the accelerometer. By setting $D = 2$, a new output will be calculated

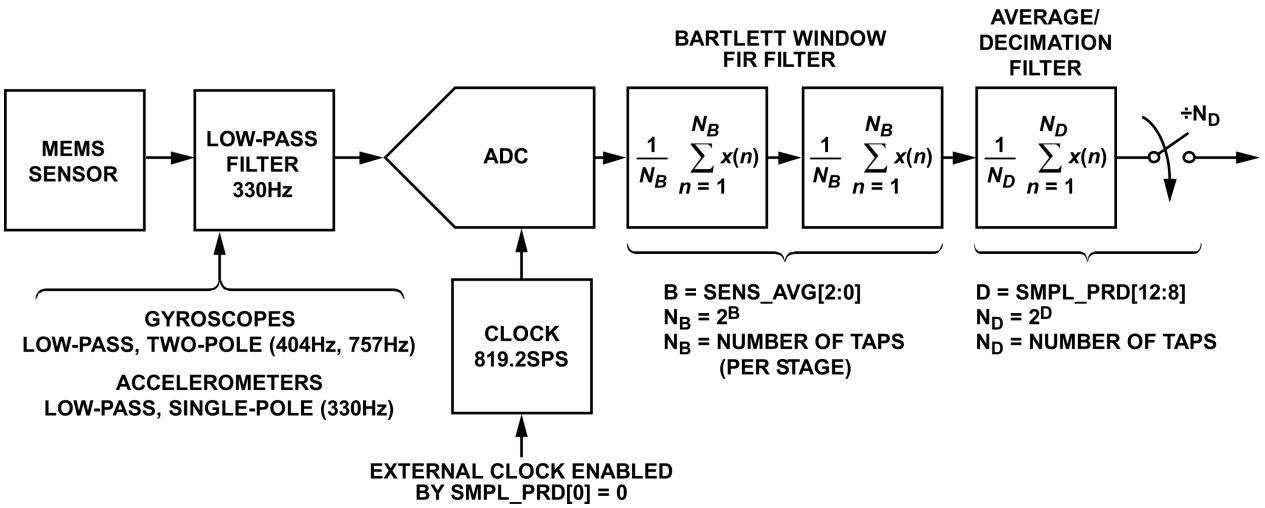

Figure 7.2: Sampling and frequency response block diagram of the ADIS16448 IMU [2].

every

$$2^2 \text{ samples} \cdot \frac{1}{819.2 \text{ samples/s}} = 4.88 \text{ ms}$$

New acceleration readings will be stored in the microcontroller asynchronous to when they are used in the velocity estimation algorithm. By updating the accelerometer reading every 4.88 ms, it can be guaranteed that a reading is not older than this.

Chapter 8

Testing and Results

This chapter will show the documented tests that were performed on the HeaveLock and its software, and the results that were obtained. In addition to the tests mentioned in this chapter, a series of minor tests were performed continuously as the HeaveLock hardware and software was incorporated and developed. These minor tests are excluded from this thesis, because the more complete tests indirectly prove that the system works on a smaller scale. As an example the CANopen library and the CAN bus needs to be configured correctly and working for the valve motor to be able to move. If this system had failed, the test of the valve actuator system in Section 8.1 could not have been performed. Consequently it can be concluded that the CANopen library and the bus functions as intended. The same reasoning can be used for the serial communication, SPI communication, and other aspects of the HeaveLock software and hardware.

8.1 Valve Actuator System

When implementation of the interface for the EPOS2 motor controller was finished, the new actuator for the HeaveLock valve had to be tested to see how well it performed. To test the system, a varying reference was sent from a Simulink diagram to HLC, which relayed the new setpoint via serial communication to the HeaveLock unit. The HeaveLock unit sent the new setpoint via CANopen to the motor controller, which executed the position controller. The new valve position was returned from the motor controller to the HeaveLock unit via CANopen, where it was sent to HLC via serial, and from HLC to Simulink via UDP.

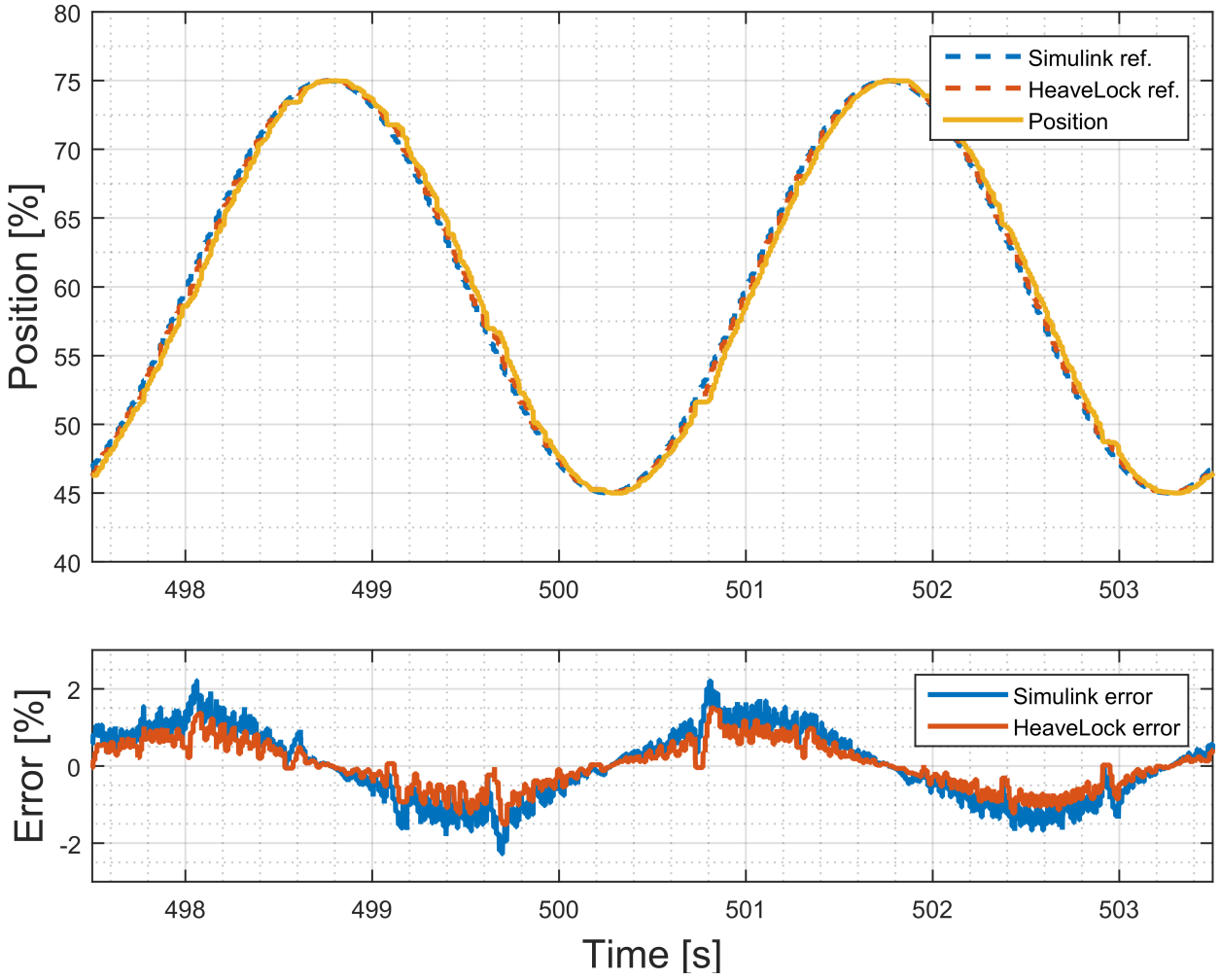

Figure 8.1: Comparison of position and reference. 3 seconds period.

Figure 8.1 shows a single sinusoid with a period of 3 seconds as the reference to follow. The result shows that the error seen from Simulink is within ±2.5 %. When the HeaveLock controller is implemented in the Arduino as an embedded system, the extra delay of transmitting the reference to the HeaveLock unit is removed, and the error due to delay of transmission is reduced. This can be seen in the error plot of the “HeaveLock error”, where the error is calculated using the reference stored in the Arduino at the time of the measured position. Using the reference stored in the Arduino, the error is within ±1.5 %.

8.2 Velocity Estimation Using Rig Data

A hardware-in-the-loop test of parameter calculation and velocity estimation was performed, sending data from the "rig data" data set used by Schaut [42] via serial to the Arduino Due. The estimated velocity that was sent in return was recorded and plotted together with the velocity reference and simulation output from Simulink. The different functions used in the parameter calculation were also timed to see how much computation time they required. By comparing the estimated velocity from Simulink and the Arduino Due, a confirmation of the implementation was achieved.

The results showed that the error between Simulink and the Arduino Due is usually within $\pm 1.0 \cdot 10^{-3}$ m/s, but with some peaks as large as $\pm 5 \cdot 10^{-2}$ m/s lasting a couple of hundred milliseconds. These peaks are a result of the changing parameters in the estimation algorithm, that are calculated every 204.8 seconds. Some delay in the calculation of new parameters was expected, as the computation power of a microcontroller is limited. In average around 120 milliseconds are used to calculate the new parameters, where the FFT calculation uses around 100 milliseconds of the total time. Note that this is the total time, including interruptions from tasks with a higher priority. The delay will remain the same each time new parameters are calculated. This is ensured by filling a new buffer each time a buffer is sent to the parameter calculation task, and so the delay does not accumulate over time. This means that the first parameter set will arrive ~204.92 s after a complete restart and initialization, while all following parameter calculations will arrive 204.8 s apart. Several implementation alternatives of some of the sub-functions were timed and tested, and the implementations that were fastest and used less flash memory were used.

8.3 Velocity Estimation Using HeaveLock and Linear Actuator

Testing of the velocity estimation on the HeaveLock was performed by using a linear actuator to move the HeaveLock unit up and down with a known position, velocity and acceleration. The linear actuator that was used was the same as mentioned in Subsection 4.1.2, i.e. the Parker Compax3 I12T11. The tests were performed by setting "go to" positions in the servo drive. To create movement similar to a sinusoidal wave, the movement to the lower and upper position

were specified with the desired position to go to, maximum velocity, maximum acceleration/deceleration and maximum jerk. The values used for the data sets can be seen in Table 8.1, and has been derived from the simple sinusoidal function shown in Equation 8.1. Logging showed that the calculated values worked quite well to produce movement that looked like sinusoidal waves. Only the jerk of the 3 s period movement had to be increased.

$$p = A \cdot sin\left(\frac{2 \cdot \pi}{T} \cdot t\right) \tag{8.1}$$

Table 8.1: Values that make up the datasets in the linear actuator.

Period	**Position Amplitude**	**Max Velocity**	**Max Acceleration**	**Max Jerk**
3 s	0.2 m	0.419 m/s	0.877 m/s^2	9.185 m/s^3
7 s	0.2 m	0.180 m/s	0.161 m/s^2	0.145 m/s^3
10 s	0.2 m	0.126 m/s	0.079 m/s^2	0.050 m/s^3

This section will show the result of three velocity estimation tests, with periods of 3, 7 and 10 s. The reason for testing the system with these oscillation periods is that this covers the specified design range of operation, namely 3-10 s periods. All tests were set up with a maximum position amplitude of 0.2 m, as the expected nominal movement in the lab will be 0.4 m, and the maximum possible movement in the lab is 0.8 m.

8.3.1 Velocity Estimation with 3 Second Periods

Figure 8.2 shows how the HeaveLock estimates the velocity of sinusoidal movement with 3 s periods. The reference that is used in the plot in Figure 8.2 is data collected from the linear actuator that moves the HeaveLock. As can be seen, the velocity estimate (red graph) closely resembles the reference velocity wave form, with the same period and minimum and maximum values. Some deviations can be recognized, especially around 0 m/s. These small peaks and troughs are falsely calculated when the direction of movement changes abruptly, i.e. going from moving in one direction, to a momentary stop, and then to move in the other direction. However, the HeaveLock quickly rectifies the velocity estimate and continues to estimate the velocity with a relatively high accuracy.

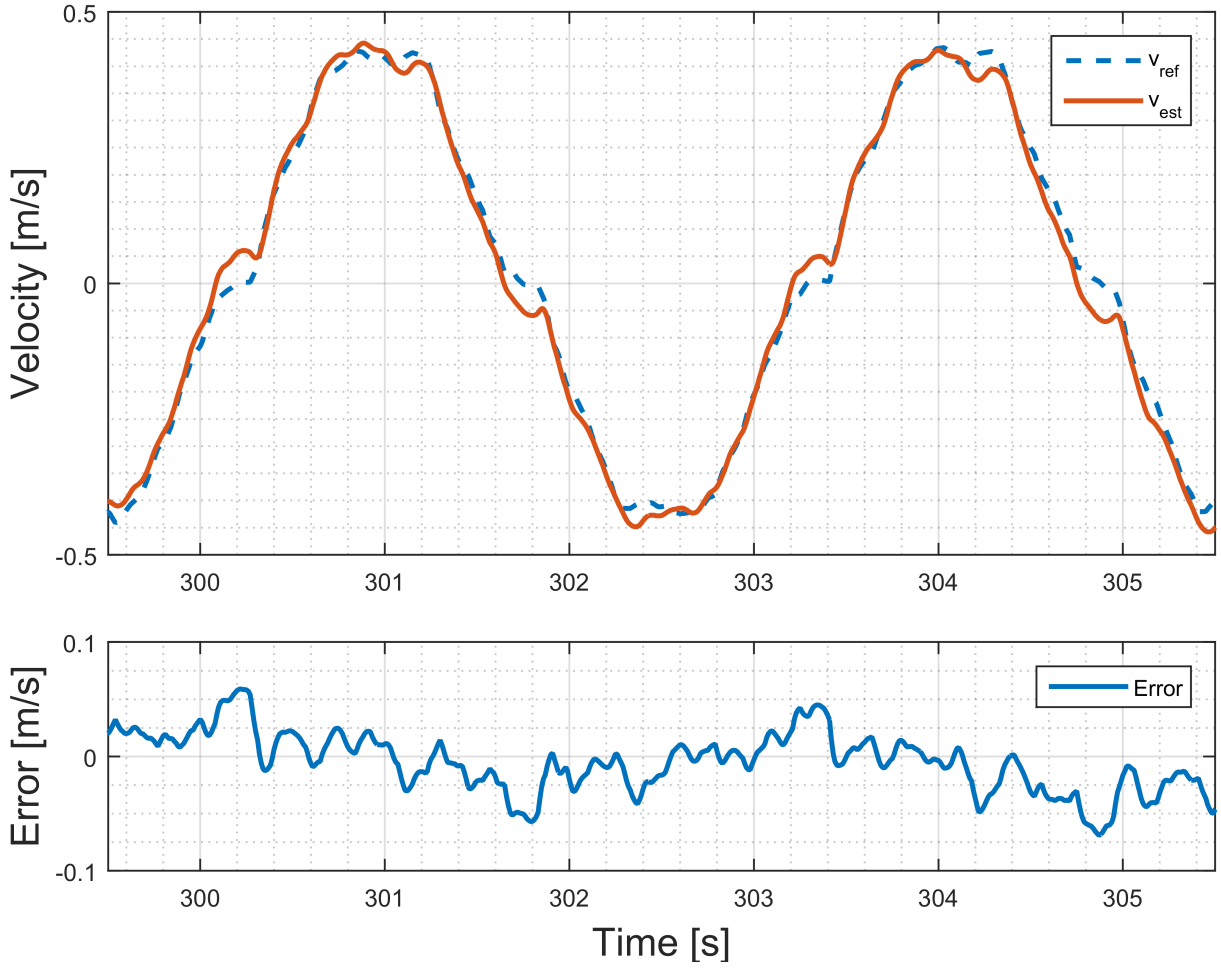

Figure 8.2: Velocity estimation of 3 s periods compared to reference velocity from linear actuator.

The error shown in the lower part of Figure 8.2 is the error between the velocity estimate and the reference velocity measured by the linear actuator. The error is usually within ±0.025 m/s with peaks extending out to ±0.075 m/s. These peaks in the error plot occurs around the point at which the HeaveLock changes direction, and is a consequence of the velocity estimate equations.

An RMS calculation of the error data collected between 250 and 450 seconds within the same data set as shown in Figure 8.2 confirmed that the the error is usually within ±0.025 m/s, with a value of $v_{\text{errorRMS}} = 0.0249$ m/s.

8.3.2 Velocity Estimation with 7 Second Periods

Figure 8.3 shows the estimated velocity for a sinusoidal movement with 7 s periods. The only difference between this experiment and the previous experiment is the period of oscillation. The result, however, is somewhat different. Longer periods with the same amplitude in position yields lower acceleration and deceleration, and so the peaks and troughs at 0 m/s in the top plot in Figure 8.3 are smaller compared to the experiment ran before. This is because the accelerometer is not subdued to quite as sudden changes in acceleration (sample to sample).

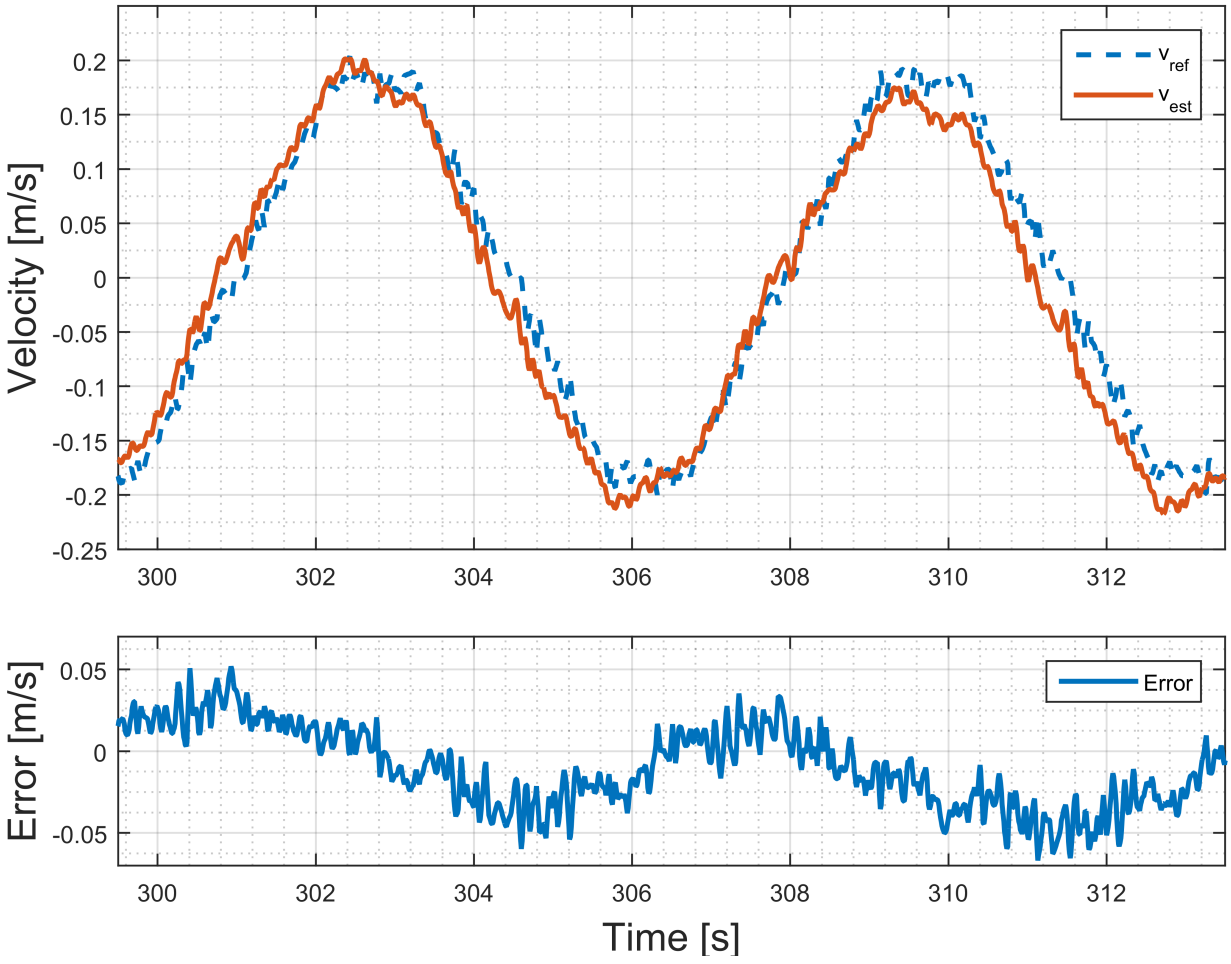

Figure 8.3: Velocity estimation of 7 s periods compared to reference velocity from linear actuator.

The error plot in Figure 8.3 shows that the maximum error during normal operation is roughly 0.05 m/s. Again, the peak error happens at the point of change in direction, but as mentioned above, the error is reduced as the period of oscillation is longer. The RMS calculation of the error

data, between 250 and 450 seconds for the same data set as displayed in Figure 8.3 yields a result of $v_{\text{errorRMS}} = 0.0214$ m/s. This suggests that slower velocities imposed on the system yields better velocity estimation by the HeaveLock.

8.3.3 Velocity Estimation with 10 Second Periods

Figure 8.4 shows the same experiment as before, only this time with 10 s periods. As the top plot show, even longer periods lead to a higher velocity estimate accuracy. And the error plot in Figure 8.4 confirms this. The peaks and troughs at the 0 m/s point is hardly distinguishable in the error plot, and what is seen is just the general error in the estimation. An RMS calculation of the error between 250 and 450 seconds results in an error of $v_{\text{errorRMS}} = 0.0133$ m/s.

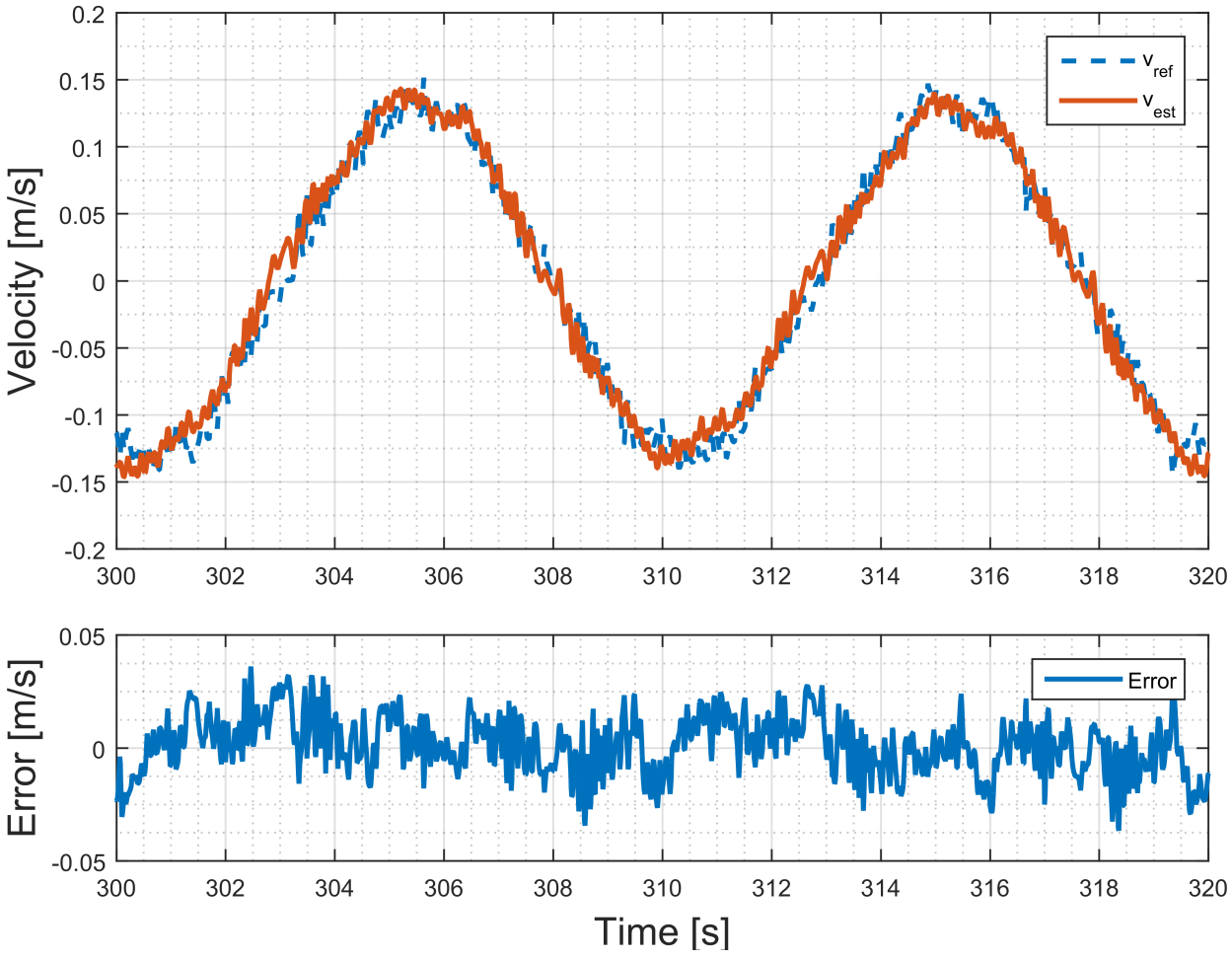

Figure 8.4: Velocity estimation of 10 s periods compared to reference velocity from linear actuator.

8.4 Testing the Analog to Digital Converter

A test of the ADC was performed by adjusting the input voltage to each channel with the help of a potentiometer. After the voltage reading had been verified for each channel a test was performed to see if reading all the channels simultaneously would produce time delays large enough to have a negative impact on the system as a whole. Before the test it was proposed to split the ADC reading over several program loops to allow for a more efficient time share of the SPI bus, used by both the ADC and the accelerometer. However, this was deemed unnecessary after the testing of the ADC was complete. The HeaveLock passed the test with time to spare for each program loop while reading all ADC channels simultaneously.

Chapter 9

Summary, Conclusion and Recommendations for Further Work

This chapter will provide a summary and conclusion of the work that has been done in this project, and also highlight some recommendations for further work and development.

9.1 Summary

Heave motion of a floating drilling rig can cause pressure fluctuations in a well, which may exceed the pressure window of the formation. A proposed solution to overcome this challenge is to install a choke valve downhole, right above the drill bit. The goal is to control the pressure, and at the same time circumvent the significant propagation delay observed if topside equipment were used. This is the idea of the HeaveLock.

At NTNU an experimental lab, called the IPT-Heave Lab, has been built to model heave motion of a drill string in a well. Over the last years, improvements and additions to the lab has been made, among these Drønnen's [15] disturbance generator and Albert's [1] topside choke valve. Controlling the flow through the annulus with the topside choke managed to reduce the pressure fluctuations by at least 46 %.

To test the principle of a downhole choke valve, a computer simulation of the system was performed by Schaut [42]. Schaut's work included suggestions for controller algorithms and heave velocity estimation equations, and the simulations showed promising results.

During the summer/fall of 2015 the lab was rebuilt to be able to test the downhole choke valve. A position controller for the valve was developed to run on a microcontroller, while Simulink handled the rest of the control algorithms on a PC. Unfortunately this system failed to operate as intended due to a physical design flaw and lack of torque in the valve actuator, making a redesign of the valve necessary.

As a step towards making the HeaveLock operate autonomously, it was decided that it should incorporate much of the logic in the control system. This led to the discretization of Schaut's velocity estimation equations, as well as an implementation of the dynamic filter parameter calculation and flow controller.

A survey was performed for the various hardware components, such as the microcontroller, accelerometer, ADC, valve actuator, and communication protocols. Hardware was selected based on functionality, cost, quality and performance, and finally assembled into one unit which now makes up the HeaveLock.

Software had to be written for the microcontroller to incorporate the logic, as well as interface the hardware. To guarantee that processing deadlines within the microcontroller were withheld, FreeRTOS was implemented as a scheduler. The software is a real-time concurrent program where functions and calculations have been divided into prioritized tasks. A main software design feature is the minimal communication scheme that was part of the initial system design. This enables the HeaveLock to be operated by a single start/stop command, enabling or disabling flow control through the drill string. However, extra communication is available to allow for logging and configuration of the HeaveLock without accessing the source code.

A GUI program was developed to enable effortless communication with the HeaveLock. This program has been named HeaveLock Control, and makes available live data from the HeaveLock, as well as the ability to move the valve and start or stop heave compensation with a single click. A purpose-built overlaying serial protocol has been implemented to both HeaveLock Control and the HeaveLock to make communication as efficient and reliable as possible.

A chapter which describes how the HeaveLock needs to be installed in the lab and which parameters that can be tuned and adjusted has been provided to ease the work of improving the characteristics of the HeaveLock.

The components that make up the HeaveLock have been tested both individually and collec-

tively, but due to the lab being occupied with other experiments it was not possible to perform the final test in the lab within the deadline of this thesis. Therefore, a pressure attenuation test has been added to the list of future work, with a recommended test procedure included in Appendix I.

9.2 Conclusion

A functioning downhole choke valve called the HeaveLock has been developed. The HeaveLock can operate with a minimum of communication with the IPT-Heave Lab control system. After an initial setup the HeaveLock only requires a start/stop heave compensation command.

Testing of the valve actuator proved that the HeaveLock controls the motor controller correctly, and that the motor controller positions the motor with insignificant position error. Some lag in the reference value was noticed, but deemed negligible for the system as a whole.

Estimating the velocity based on accelerometer data from a real oil rig showed that the discretized heave filter equations are correct. A small error was observed, but the majority of the error occurs at parameter changes in the filter. This was expected and the test was considered successful.

The velocity estimation tests showed that the velocity estimation works, and that the HeaveLock can estimate the velocity of oscillation periods varying from 3 s to 10 s. The error of these estimations were in the range of 2.49 cm/s to 1.33 cm/s, decreasing as the oscillation periods increased. A specification for the maximum error allowed has not been presented, so whether or not the estimation error is large enough to affect the HeaveLock's ability to reduce pressure fluctuations in the lab is unknown. Pressure attenuation tests must be performed in the lab to determine the performance of the HeaveLock, but it is assumed that the error of the velocity estimation is small enough to yield good results. The largest deviations in the estimate occurred when the HeaveLock changed direction, and these periodic deviations are visible at shorter periods. However, when the system was subjected to longer periods, the periodic deviations were hardly distinguishable from the general error. As the periods increased, also the general error was reduced.

Testing of the analog to digital converter proved that it operated correctly. As this was the

final test of the entire system it furthermore proved that the HeaveLock has time enough to complete all its tasks for each program loop. It also has time to spare for tasks that require more than one program loop to finish, e.g. calculation of filter parameters based on the dominant wave amplitude and frequency.

With the exception of the pressure attenuation test of the HeaveLock in the IPT-Heave Lab, all objectives of this thesis have been met.

9.3 Recommendations for Further Work

What follows is a prioritized list of proposed future work:

1. Test the HeaveLock system in the IPT-Heave Lab. A test procedure has been prepared, see Appendix I.

2. Figure out why an optimal cutoff frequency is not calculated correctly and improve the optimization equations.

3. Consider replacing the accelerometer if the velocity estimation has to be improved. A new accelerometer should have:

    - A smaller range, i.e. better resolution in the range of operation for this system.
    - The ability to configure the sample period easily, and thereby enable optimization of sample times in the HeaveLock software.

4. Implement dynamic removal of the gravity component from the accelerometer data. Gravity is now removed by subtracting a constant from the measurement, and will not work correctly for movement in a horizontal direction.

5. Optimize the parameter calculation if it takes to much time to complete.

    - Update the CMSIS-DSP library to the newest version.
    - Exchange the outdated “arm_rfft_f32” function with the successor “arm_rfft_fast_f32” function, or change the buffer data types from floating point to fixed point to allow

for faster calculation. This involves changing the CMSIS functions from "…_f32"-type to "…_q31".

Appendix A

P&ID

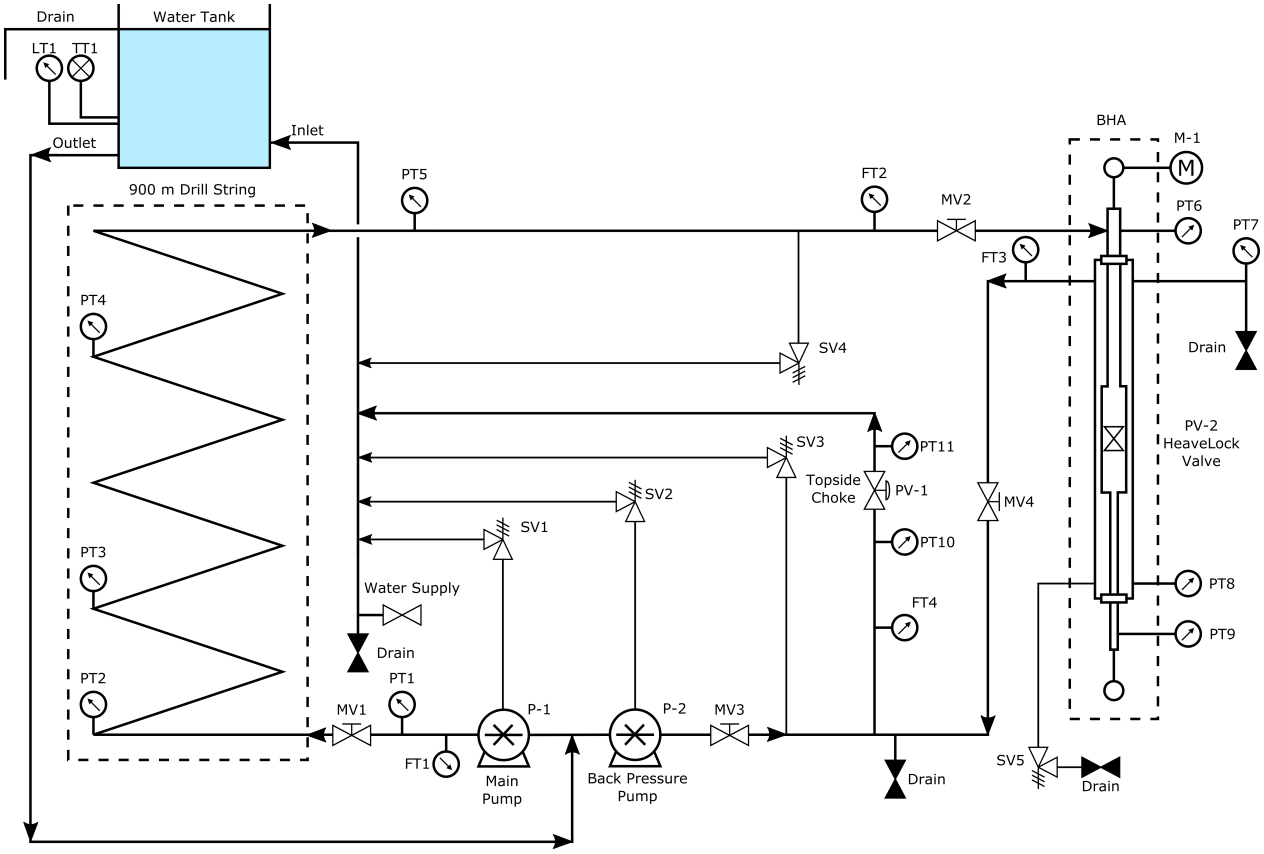

Figure A.1: P&ID of the IPT-Heave Lab, adapted from [14] and [20].

Appendix B

CAN Shield Schematic

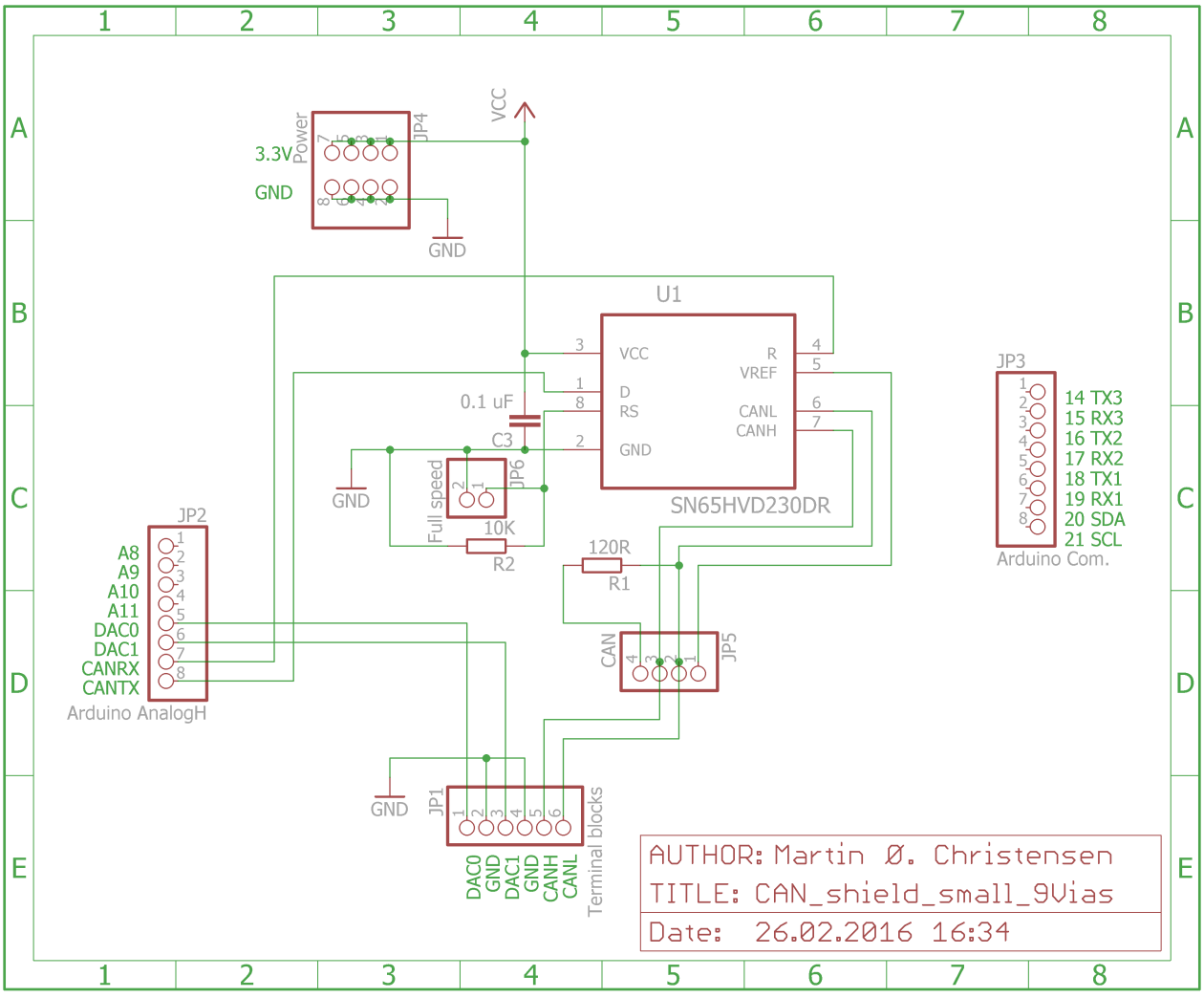

Figure B.1: CAN Shield Schematic.

Appendix C

HeaveLock Software File Overview

ADIS16448.h / ADIS16448.cpp

Number of Lines: 125 / 206

Used By: HeaveLockSW.ino

Includes: SPI.h, FreeRTOS_ARM.h

Description: Includes functions to configure and use the accelerometer.

Buffer.h / Buffer.cpp

Number of Lines: 34 / 40

Used By: HeaveLockSW.ino, ParamEstimation.cpp, Hamming.h

Includes: N/A

Description: Makes available a buffer for storing data.

CANopen.h / CANopen.cpp

Number of Lines: 99 / 268

Used By: EPOS.h

Includes: due_can.h

Description: Includes functions to send and receive CANopen messages.

EPOS.h / EPOS.cpp

Number of Lines: 102 / 179

Used By: HLController.h

Includes: CANopen.h

Description: Provides functionality for communication with the EPOS motor controller.

ExtendedADCShield.h / ExtendedADCShield.cpp

Number of Lines: 57 / 140

Used By: HLController.h

Includes: FreeRTOSConfig.h, FreeRTOS_ARM.h, SPI.h

Description: Makes available functions for reading signals from analog devices connected to the ADC.

FilterParam.h

Number of Lines: 20

Used By: HeaveFilter.cpp, HeaveLockSW.ino, HLController.h, HLSerial.cpp, ParamEstimation.cpp

Includes: N/A

Description: A type definition for a “struct” that contains the filter parameters.

FreeRTOSConfig.h

Number of Lines: 201

Used By: HeaveLockSW.ino, HLController.h, TaskParam.h, ExtendedADCShield.h, ParamEstimation.cpp

Includes: stdint.h

Description: Configuration file for FreeRTOS. Contains information of tick rate, stack size, priorities etc.

Hamming.h

Number of Lines: 444

Used By: ParamEstimation.cpp

Includes: Buffer.h

Description: Look-up tables of different lengths containing Hamming window factors.

HeaveFilter.h / HeaveFilter.cpp

Number of Lines: 26 / 29

Used By: HeaveLockSW.ino

Includes: FilterParam.h

Description: Makes available the heave filter and phase correction functions for calculating estimated velocity based on acceleration data.

HeaveLockSW.ino

Number of Lines: 286

Used By: N/A

Includes: FreeRTOSConfig.h, FreeRTOS_ARM.h, ADIS16448.h, Buffer.h, FilterParam.h, HeaveFilter.h, HLController.h, HLSerial.h, ParamEstimation.h, TaskParam.h

Description: Main file. Starts tasks and scheduler. All tasks but the calculateFilterParametersTask lives here. See Section 5.2 for a task overview.

HLController.h / HLController.cpp

Number of Lines: 125 / 286

Used By: HeaveLockSW.ino

Includes: FreeRTOSConfig.h, FreeRTOS_ARM.h, EPOS.h, ExtendedADCShield.h, FilterParam.h, HLSerial.h, TelemetryData.h

Description: Contains functions for calculating desired flow through BHA, and desired valve opening. The HeaveLock controller and the inverse choke characteristic are located here.

HLSerial.h / HLSerial.cpp

Number of Lines: 110 / 149

Used By: HeaveLockSW.ino, HLController.h, ParamEstimation.cpp

Includes: FilterParam.h, TelemetryData.h

Description: Contains serial command definitions and functions for sending and receiving HeaveLock messages over a serial connection.

ParamEstimation.h / ParamEstimation.cpp

Number of Lines: 27 / 142

Used By: HeaveLockSW.ino

Includes: arm_math.h, Buffer.h, FilterParam.h, FreeRTOSConfig.h, FreeRTOS_ARM.h, Hamming.h, HLSerial.h, TaskParam.h

Description: The calculateFilterParametersTask lives here. Also, all functions and equations related to the sea state estimation and parameter calculation are located here.

TaskParam.h

Number of Lines: 58

Used By: ParamEstimation.cpp, HeaveLockSW.ino

Includes: FreeRTOSConfig.h, FreeRTOS_ARM.h

Description: Defines a set of data types for passing arguments to tasks at task generation. The majority of the HeaveLock's tunable parameters are defined here.

TelemetryData.h

Number of Lines: 17

Used By: HLController.h, HLSerial.cpp

Includes: N/A

Description: A type definition that is used for collecting and transmitting telemetry data to HLC.

Appendix D

EPOS2 CANopen Object Dictionary

Some of the objects in the CANopen object dictionary must be changed from their default values in the setup of the EPOS2 motor controller. Table D.1 lists the controller tuning objects for the position controller. The values of these objects were found using the "Regulation Tuning wizard" in EPOS Studio, running the auto tuning procedure. Table D.2 lists the objects that must be changed to fit the motor and encoder that is used, and to set up correct CANopen communication with the HeaveLock.

Table D.1: Part of the object dictionary regarding controller tuning.

Index	Sub	Name	Default Value	New Value
0x60F6	01	Current Regulator P-Gain	300	167
0x60F6	02	Current Regulator I-Gain	100	56
0x60F9	01	Speed Regulator P-Gain	1000	2506
0x60F9	02	Speed Regulator I-Gain	100	384
0x60F9	05	Acc. Feedfwd. Factor in Speed Reg.	0	325
0x60FB	01	Position Regulator P-Gain	150	576
0x60FB	02	Position Regulator I-Gain	10	1884
0x60FB	03	Position Regulator D-Gain	200	940
0x60FB	05	Acc. Feedfwd. Factor in Position Reg.	0	325

Table D.2: Part of the object dictionary regarding communication and motor setup.

Index	Sub	Name	Default Value	New Value
0x1800	01	COB-ID used by TxPDO 1	Node ID+0x40000180	0x00000181
0x1801	01	COB-ID used by TxPDO 2	Node ID+0xC0000280	0x00000281
0x1802	01	COB-ID used by TxPDO 3	Node ID+0xC0000380	0x00000381
0x1802	02	Transmission Type	0xFF	0x01
0x1803	01	COB-ID used by TxPDO 4	Node ID+0xC0000480	0x00000481
0x1803	02	Transmission Type	0xFF	0x01
0x1A03	01	1st Mapped Object in TxPDO 4	0x60410010	0x60610008
0x2000		Node ID	Node ID (DIP switch)	0x01
0x2079	05	Configuration of Digital Output5	11	255
0x2080		Current Threshold for Homing Mode	500	5000
0x2081		Home Position	0	16000
0x2210	02	Position Sensor Type	1	4
0x2210	04	Position Sensor Polarity	0x0000	0x0004
0x2211	02	SSI Encoder Number of Data Bits	0x0C0D	0x000C
0x2211	04	SSI Encoder Encoding Type	0x0000	0x0001
0x607C		Home Offset	0	2000
0x607F		Max Profile Velocity	25000	10000
0x6098		Homing Method	7	-3
0x609A		Homing Acceleration	1000	50
0x60C5		Max Acceleration	4294967295	2000
0x6402		Motor Type	10	1
0x6410	04	Maximal Motor Speed	25000	11300
0x6410	05	Thermal Time Constant Winding	40	440

Appendix E

Modifications Regarding the CMSIS Library

Currently version 1.0.10 of the CMSIS-DSP library is included in the distribution of the Arduino board software, however it is not possible to use it without some modifications. The software library is not included in the build process, and without any modifications there will be an error when the project is built if some of the functions in the library have been used. The following modifications will make the linker find the CMSIS library:

The file `libarm_cortexM3l_math.a` must be copied from
`C:\Users\<user name>\AppData\Local\arduino15\packages\arduino\hardware\sam\1.6.7\system\CMSIS\CMSIS\Lib\GCC`
to
`C:\Users\<user name>\AppData\Local\arduino15\packages\arduino\hardware\sam\1.6.7\variants\arduino_due_x`

The file `platform.txt` also has to be modified. It is located in the folder
`C:\Users\<user name>\AppData\Local\arduino15\packages\arduino\hardware\sam\1.6.7`
`"build.variant.path/libarm_cortexM3l_math.a"` has to be inserted right after `"build.variant.path/build.variant_system_lib"` in the "Combine gc-sections, archives, and objects"-section of the file.

Appendix F

HLC Error Messages

Message HLC: Tail not found when expected. Command used in missed message: xx.

Cause Something went wrong with the transmission of serial messages.

Solution If this happens regularly an unknown bug may exist. If it happens from time to time it may be interference on the serial connection.

Message HLC: Tail not found when expected. Command used in missed message: 0. Baud rate may be wrong. Change baud rate and reconnect.

Cause Most likely wrong baud rate selected.

Solution Try changing the baud rate to 238 636.

Message HLC: Serialread timed out locally. Command used in missed message: xx.

Cause The Windows scheduler may override all processes running on a computer. If this happens when a serial read is to be executed this may interfere with the communication between HLC and HeaveLock.

Solution If this interferes with data logging, try to reduce CPU load.

Message Creation problem.

Cause Error creating tasks in the HeaveLock program.

Solution Check source code. Has it been changed?

Message Did not receive accData successfully.

Cause Wires may have been disconnected. Accelerometer may be broken. There may be a FreeRTOS queue problem.

Solution Check wires and connections. Reboot and try again. If that fails, check the accelerometer for faults.

Message Could not send accBuffer to queue.

Cause Something is wrong with the FreeRTOS queues in the HeaveLock program.

Solution Check source code and documentation.

Message readAcc timed out...

Cause Could be caused by wrong task initialization (e.g. wrong priority assignment). Could also be wrong FreeRTOS config, or other unknown FreeRTOS bugs related to queues.

Solution Try a restart. If the same problem reoccurs, check source code and documentation.

Message Did not receive v_hat successfully.

Cause Could be caused by wrong task initialization (e.g. wrong priority assignment). Could also be wrong FreeRTOS config, or other unknown FreeRTOS bugs related to queues.

Solution Try a restart. If the same problem occurs, check source code and documentation.

Message Incomplete message received.

Cause Decoding of a received message failed. If the serial buffer of the HeaveLock runs full, this error will be sent very frequently.

Solution Reduce serial traffic.

Message Serial transmission timed out.

Cause Decoding a message took to long. If it happens frequently this may be because of overload of the CPU.

Solution If no changes has been made to the source code, this may be an unknown bug. Check source code.

Message Invalid setpoint.

Cause An invalid setpoint was sent.

Solution Setpoints has to be between 0 % and 100 %.

Message Unknown motor controller state.

Cause Something is awry with the motor controller. Check datasheet.

Solution Try rebooting everything. If the error persists and/or resurfaces check the datasheet for the motor controller. Connect the motor controller to a PC, open EPOS studio, try to recreate the error and look for changes in the registers.

Message Failed to initialize motor controller.

Cause CANopen communication may not be working correctly, or motor controller is not configured correctly

Solution Check the communication wires between HeaveLock and motor controller. Make sure all systems have power. If none of these suggestions remedies the problem check the motor controller configuration according to Appendix D.

Message Invalid command received.

Cause An invalid command was sent to HeaveLock from HLC, or the serial data decoder in HeaveLock fails to decode data correctly. May also happen occasionally if the serial buffer is full in the HeaveLock.

Solution If no changes has been made to the source code, and still this error persists, an unknown bug has surfaced. Otherwise, reduce the amount of messages sent, so that no more than one message every 10 ms is sent to the HeaveLock.

Message Homing needs to be executed.

Cause The valve is not allowed to move before a homing run has been executed for the valve motor.

Solution Perform homing via HLC.

Message Failed to receive new acceleration data buffer.

Cause Could be caused by wrong task initialization (e.g. wrong priority assignment). Could also be wrong FreeRTOS configuration, or other unknown FreeRTOS bugs related to queues.

Solution Try a restart. If the same problem occurs, check source code and documentation.

Message rfft_init failed during parameter calculation.

Cause Something went wrong while initializing the FFT calculations.

Solution Check source code, has anything been changed recently?

Appendix G

I/O-List

IO-LIST NI DAQ - Lab computer								DATE 24.05.2016		Name EG
PC-I/O				FIELD				Scale in Simulink		
DI	AI	DO	AO	TAG	Description	Product code	Signal type	Min	Max	Unit
	ai0:0			FT-1	Flow, upstream coil	CAM-9A-100003532	(-)10 V - 10 V	0	20	l/min
	ai0:1			FT-2	Flow, drillbit inlet	Optiflux 2000F	4-20 mA	0	30	l/min
	ai0:2			FT-3	Flow, annulus outlet	Optiflux 2000F	4-20 mA	0	30	l/min
	ai0:3			FT-4	Flow, topside choke	Optiflux 2000F	4-20 mA	0	30	l/min
	ai0:4			PT-1	Pressure, main pump outlet	PTX5072.70G	4-20 mA	0	70	bar
	ai0:5			PT-2	Pressure, upstream coil	PTX5072.70G	4-20 mA	0	70	bar
	ai0:6			PT-3	Pressure, coil	PTX5072.70G	4-20 mA	0	70	bar
	ai0:7			PT-4	Pressure, downstream coil	PTX5072.70G	4-20 mA	0	70	bar
	ai0:8			PT-5	Pressure, coil outlet	PTX5072.70G	4-20 mA	0	70	bar
	ai0:9			PT-6	Pressure, top of BHA	PTX5072.70G	4-20 mA	0	70	bar
	ai0:10			PT-7	Pressure, annulus outlet	PTX5072.25G	4-20 mA	0	25	bar
	ai0:11			PT-8	Pressure, downstream drill bit	PTX5072.25G	4-20 mA	0	25	bar
	ai0:12			PT-9	Pressure, downstream HeaveLock	PTX5072.25G	4-20 mA	0	25	bar
	ai0:13			PT-10	Pressure, upstream topside choke	PTX5072.25G	4-20 mA	0	25	bar
	ai0:14			PT-11	Pressure, downstream topside choke		4-20 mA	0	25	bar
	ai0:15			LT-1	Level, water tank		4-20 mA	0	100	%
	ai0:19			ZT-1	Position, heave motor		2-10 V	0	100	%
	ai0:20			ZT-3	Position, topside choke			0	100	%
	ai0:21			TT-1	Temperature, water tank		(-)10 V - 10 V	0	100	°C
			ao0:0	P-1	Main pump	MC07B0055-5A3-4-00	0-10 V	0	100	%
			ao0:2	M-1	Heave motor	Lenze VSD	(-)10 V - 10 V	0	100	%

Figure G.1: List of all I/O for the IPT-Heave Lab.

Appendix H

HeaveLock Circuit Diagram

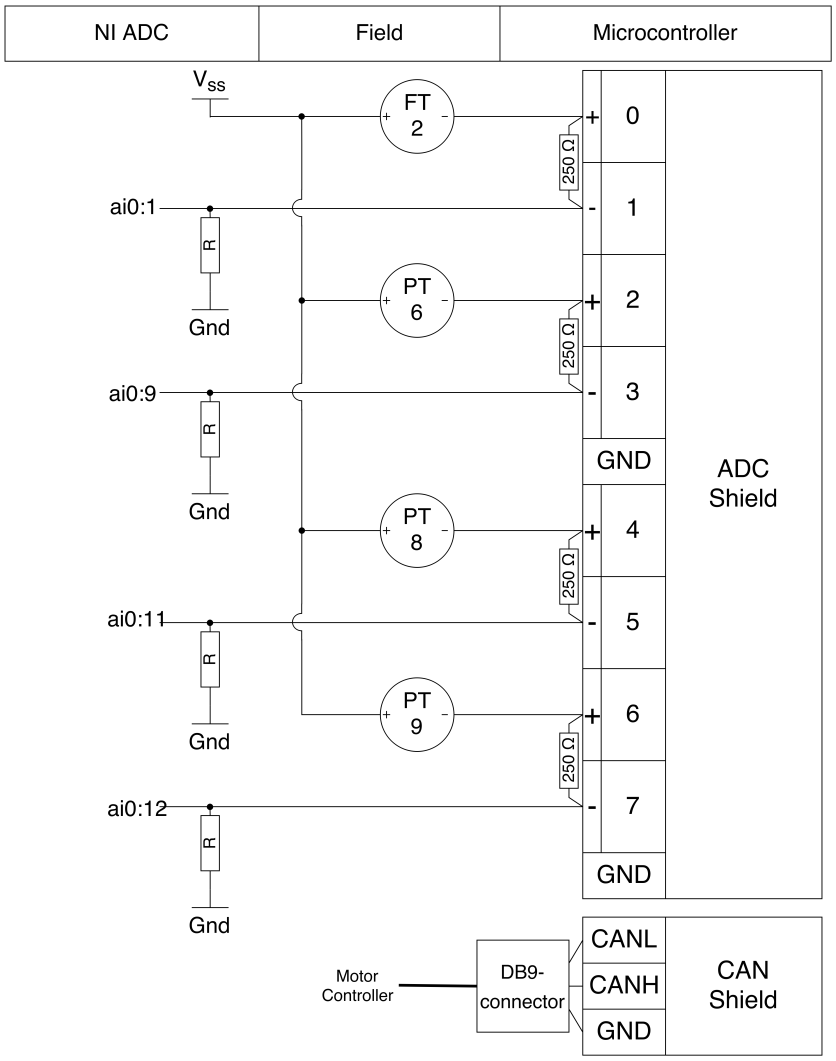

Figure H.1: Instrumentation and motor controller connection to the HeaveLock.

Appendix I

Proposed Initial Wet Test for the HeaveLock

An installation and performance test should be executed for the HeaveLock system in the IPT-Heave Lab. The goal of this test is to check that the behaviour of the HeaveLock is correct, and that it works as intended. What follows is a description of a proposed test that should be run initially after installation. Specifically one should look for discrepancy in the HeaveLock controller implemented in the HeaveLock compared to an implementation in Simulink, to verify that the HeaveLock controller is implemented correctly. Also, one should look at the HeaveLock's ability to control the downhole pressure, and compare the results to earlier tests with topside equipment.

1. Make sure that the HeaveLock is powered up, and that it is at rest at start-up (~2 seconds).
2. Select the simple feed forward HeaveLock controller in the control system (Simulink).
3. Energize the lab and start flow through the BHA.
4. Excite the system using the heave disturbance generator developed by Drønnen [15].
5. Make sure that heave compensation is deactivated for the HeaveLock (use HLC).
6. Set the valve opening manually to 100 % using HLC.
7. Log the downhole pressure for a specified time / number of periods.
8. Set the pump flow rate in HLC to the actual flow rate in the lab.

9. Start heave compensation in the HeaveLock via HLC.

10. Log the following values while heave compensation is active:.

    - Log the HeaveLock's desired valve opening in Simulink via HLC.
    - Log the desired valve opening from the HeaveLock controller implemented in Simulink.
    - Log the downhole pressure.

11. Once a reasonable time or number of periods has passed:

    - Compare the desired valve opening of the HeaveLock with the desired valve opening calculated by the topside control system.
    - Verify that these desired valve openings match with a certain degree of accuracy. Note that time delay might come into play here because of transmission time.
    - Compare the BHA pressure measurements (without and with heave compensation).

If a discrepancy is found between the two desired valve openings, this will indicate that the implementation of the HeaveLock controller and/or the inverse choke characteristic is different for the topside control system (Simulink) and the HeaveLock software. The source code in both Simulink and the HeaveLock software should then be compared to the HeaveLock controller in Equation 5.1 and the inverse choke characteristic in Equation 5.5.

Once the implementation of the HeaveLock controller has been verified, the result of the pressure attenuation test should be compared to the work of Albert [1, Section 7.4]. Hopefully the HeaveLock can outperform the topside MPD system used by Albert [1]. A failure to attenuate pressure fluctuations might indicate that the velocity estimation is not good enough, or that the simple feed forward controller used to determine the desired flow through BHA, Equation 5.1, needs to be replaced.

References

[1] Albert, A. (2013). Disturbance Attenutation in Managed Pressure Drilling. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

[2] Analog Devices (2015). ADIS16448 Datasheet. [Online; Accessed February 8, 2016] http://www.analog.com/media/en/technical-documentation/data-sheets/ADIS16448.pdf.

[3] Arduino LLC (2016a). A Brief Introduction to the Serial Peripheral Interface (SPI). [Online; Accessed May 4, 2016] https://www.arduino.cc/en/reference/SPI.

[4] Arduino LLC (2016b). Arduino Nano. [Online; Accessed February 12, 2016] https://www.arduino.cc/en/Main/ArduinoBoardNano.

[5] Arduino LLC (2016c). Download the Arduino Software. [Online; Accessed May 5, 2016] https://www.arduino.cc/en/Main/Software.

[6] Arduino S.r.l (2016). Arduino Due Board. [Online; Accessed May 4, 2016] http://www.arduino.org/images/products/ArduinoDue3_3_flat.jpg.

[7] ARM Ltd. (2015). CMSIS-DSP Software Library - RealFFT. [Online; Accessed May 9, 2016] https://www.keil.com/pack/doc/CMSIS/DSP/html/group___real_f_f_t.html#ga10717ee326bf50832ef1c25b85a23068.

[8] Atlassian (2016a). BitBucket. [Online; Accessed May 4, 2016] https://bitbucket.org/.

[9] Atlassian (2016b). SourceTree. [Online; Accessed May 4, 2016] https://www.sourcetreeapp.com/.

[10] Atmel (2015). SAM3X/SAM3A Series Datasheet. [Online; Accessed March 04, 2016] http://www.atmel.com/images/atmel-11057-32-bit-cortex-m3-microcontroller-sam3x-sam3a_datasheet.pdf.

[11] CAN in Automation (CiA) e. V. (2011). CiA 301: CANopen application layer and communication profile. [Online; Accessed February 8, 2016] http://www.can-cia.org/standardization/specifications/.

[12] Caro, D. (2009). *Automation Network Selection: A Reference Manual*. International Society of Automation, 2nd edition.

[13] Chong, J. J. (2015). ADIS16448 Arduino Demo Library. [Online; Accessed May 25, 2016] https://github.com/juchong/ADIS16448-Arduino-Demo.

[14] Christensen, M. Ø. (2015). Analysis and Testing of the HeaveLock and its Control System. Project report, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

[15] Drønnen, R. T. (2013). Generering av Forstyrrelse i et Eksperimentelt Lab-oppsett for Managed Pressure Drilling. Master's thesis, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

[16] Endevco / Meggitt Sensing Systems (2012). Steps to selecting the right accelerometer. [Online; Accessed January 15, 2016] https://www.endevco.com/news/newsletters/2012_07/tp327.pdf.

[17] Git (2016). Git Home Page. [Online; Accessed May 4, 2016] https://git-scm.com/.

[18] Godhavn, J.-M. (1998). Adaptive Tuning of Heave Filter in Motion Sensor. In *OCEANS'98 Conference Proceedings*, volume 1, pages 174–178. IEEE.

[19] Greiman, B. (2014). FreeRTOS port for Arduino. [Online; Accessed May 25, 2016] https://github.com/greiman/FreeRTOS-Arduino.

[20] Gundersen, E. (2015). Development, Analysis and Testing of the HeaveLock and its Control System. Project report, Norwegian University of Science and Technology, Department of Engineering Cybernetics.

[21] Hannegan, D. M. (2006). Case Studies-Offshore Managed Pressure Drilling. In *SPE Annual Technical Conference and Exhibition*. Society of Petroleum Engineers.

[22] Harmonic Drive AG (2014). HA-680 Controller for Mini Servo Actuators. [Online; accessed January 27, 2016] http://harmonicdrive.de/en/products/servo-products/controller/ha-680.html.

[23] InvenSense (2016). 6-Axis Gyro+Accel: MPU-6050. [Online; Accessed February 12, 2016] http://store.invensense.com/ProductDetail/MPU6050-InvenSense-Inc/422200/.

[24] ISO/IEC (1989). ISO/IEC 7498-4 Information processing systems - Open Systems Interconnection - Basic Reference Model - Part 4: Management framework.

[25] Kidder, C. (2013). CAN Library for Arduino Due. [Online; Accessed May 25, 2016] https://github.com/collin80/due_can.

[26] Klotz, C., Bond, P. R., Wassermann, I., Priegnitz, S., et al. (2008). A New Mud Pulse Telemetry System for Enhanced MWD/LWD Applications. In *IADC/SPE Drilling Conference*. Society of Petroleum Engineers.

[27] Kongsberg Maritime (2015). Kongsberg Seatex MRU5+ Datasheet. [Online; Accessed January 18, 2016] https://www.km.kongsberg.com/ks/web/nokbg0397.nsf/AllWeb/A835CAA3D6E43C6BC12575AE004AA55B/$file/Datasheet_mru5plus_MkII_march15.pdf.

[28] Linear Technology (2004). LTC1857/LTC1858/LTC1859. [Online; Accessed February 28, 2016] http://mayhewlabs.com/media/LTC185x%20Datasheet.pdf.

[29] Mathas, C. (2013). What You Need to Know to Choose an Accelerometer. [Online; Accessed January 15, 2016]
http://www.digikey.com/en/articles/techzone/2013/oct/what-you-need-to-know-to-choose-an-accelerometer.

[30] MathWorks (2015). Simulation and Model-Based Design. [Online; Accessed December 16, 2015]
http://se.mathworks.com/products/simulink/.

[31] maxon motor ag (2013). EPOS2 Communication Guide. [Online; Accessed April 7, 2016]
http://www.maxonmotor.com/medias/sys_master/8806425067550/EPOS2-Communication-Guide-En.pdf.

[32] maxon motor ag (2016a). EPOS Studio (direct download). [Online; Accessed May 5, 2016]
http://www.maxonmotor.com/medias/sys_master/root/8818445451294/epos2-setup.zip.

[33] maxon motor ag (2016b). EPOS2 Motor Controller. [Online; Accessed May 31, 2016]
http://www.maxonmotor.com/medias/sys_master/root/8797301243934/PRODUKTBILD-EPOS-2-50-5-347717-Detail.jpg.

[34] maxon motor ag (2016c). MAXPOS Motor Controller. [Online; Accessed January 27, 2016]
http://www.maxonmotor.co.uk/maxon/view/content/MAXPOS-Detailsite.

[35] Mayhew Labs (2015). Extended ADC Shield. [Online; Accessed February 28, 2016]
http://mayhewlabs.com/products/extended-adc-shield.

[36] Microsoft (2016). Visual Studio. [Online; Accessed May 4, 2016]
https://www.visualstudio.com/.

[37] Parker Hannifin (2005). Operating Instructions Compax3 I12T11. [Online; Accessed May 16, 2016]
http://www.parkermotion.com/manuals/Hauser/Compax3/C3I12T11_Aug-2005.pdf.

[38] Parker Hannifin (2015a). LCB SERIES - LCB040 Belt Driven, Slider Bearing Rodless Linear Actuator. [Online; Accessed May 16, 2016]
http://ph.parker.com/us/en/lcb-series-lcb040-belt-driven-slider-bearing-rodless-linear-actuator.

[39] Parker Hannifin (2015b). SMH-SERIES Low-Inertia Brushless Servo Motors (For Use With COMPAX3 Servo Drives). [Online; Accessed May 16, 2016]
http://ph.parker.com/us/en/smh-series-low-inertia-brushless-servo-motors-for-use-with-compax3-servo-drives.

[40] Real Time Engineers Ltd. (2016). FreeRTOS Website. [Online; Accessed February 12, 2016]
http://www.freertos.org/.

[41] Richter, M., Schneider, K., Walser, D., and Sawodny, O. (2014). Real-Time Heave Motion Estimation using Adaptive Filtering Techniques. In *Proc. of the 19th IFAC World Congress*, pages 10119–10125.

[42] Schaut, S. (2015). Modeling, Estimation and Control for an experimental Lab Facility for Drilling. Master's thesis, University of Stuttgart, Institude for Systems Theory and Automatic Control.

[43] SparkFun Electronics (2013). Serial Pripheral Interface (SPI). [Online; Accessed May 4, 2016]
https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi.

[44] Texas Instruments (2015). SN65HVD23x 3.3V CAN Bus Transceivers. [Online; Accessed March 6, 2016]
http://www.ti.com/lit/ds/symlink/sn65hvd230.pdf.

[45] The Underbalanced Operations & Managed Pressure Drilling Committee of the International Association of Drilling Contractors (2011). The UBO & MPD Glossary. [Online; accessed December 9, 2015]
http://www.iadc.org/wp-content/uploads/UBO-MPD-Glossary-Dec11.pdf.

[46] Visual Micro (2016). Arduino Plugin for Visual Studio. [Online; Accessed May 5, 2016] http://www.visualmicro.com/.

[47] Westermo Teleindustri AB (2004). Industrial Data Communication - Theoretical and General Applications. [Online; Accessed December 14, 2015] http://ftc.beijer.se/files/C125728B003AF839/7D40AB0A54C6D7F2C1257B64004B49C4/westermo_theoretical_handbook_v5_0_en.pdf.

[48] Xsens (2015). MTi 100-series. [Online; Accessed January 18, 2016] https://www.xsens.com/download/pdf/documentation/mti-100/mti-100-series.pdf.