A hydraulics simulation system

Using a Hardware In the Loop approach

V.G. Zaccà
J.M.S. van Rijn

A hydraulics simulation system

Using a Hardware In the Loop approach

by

V.G. Zaccà
J.M.S. van Rijn

to obtain the degree of Bachelor of Science
at the Delft University of Technology,

Student number Gabriele Zacca:	4368967	
Student number Joey van Rijn:	4321413	
Project duration:	April 24, 2017 – June 21, 2017	
Thesis committee:	dr. ir. N.P. van der Meijs,	TU Delft, supervisor
	Prof. dr. P. Palensky,	TU Delft
	Dr. F. Uysal,	TU Delft
	A. Ramaker,	HedoN

This thesis is confidential and cannot be made public until April 21, 2020.

TU Delft

Executive Summary

Huisman is an industry leader in the design and manufacturing of heavy construction equipment. To achieve high acceleration controlled operations for large cranes, secondary hydraulic systems are designed. The control system for these secondary hydraulic plants is very complex and designed by Hedon. HedoN is a high-tech electronic development and production company that tasked a group of six TU Delft Electrical Engineering bachelor students to design a Hardware-In-The-Loop (HIL) simulation system to connect to a hydraulic motor controller (HMC), as a cheaper alternative to real hydraulic machinery. This simulation system consist of a central Hydraulic Simulation Unit (HSU) that runs a hydraulics simulation and controls various electrical I/O hardware modules (with a local microprocessor and Ethernet port) that simulates hydraulic actuators and various sensors. This hardware I/O modules of the simulation system can be connected to the hydraulic and mechanical I/O of the HMC, and can be used by a hydraulic commissioning engineers as a tool to configure the HMC for various complex hydraulic plants or as a teaching tool for junior commissioning engineers.

The HSU is designed as the master of all the I/O hardware module connected to the Ethernet switch. The HSU is divided into four different components: The Ethernet communication module, the Ethernet packet data structurer, the analog IO interpreter/translator and the simulation engine itself. The latter component simulates all the hydraulic and mechanical dynamics of the plant and is the only application specific component of the overall simulation system.

The simulation system architecture has been designed and implemented on a microprocessor running Linux. The implementation has been partially verified by testing each component separately and by testing the integrated system. Due to the lack of hardware I/O modules the simulation system has not been tested in combination with the HMC.

This thesis describes the system architecture and implementation of the hydraulic simulation system that allows for a modular and flexible simulation system that can be easily expanded with additional custom I/O hardware modules or be applied in a different industry such as the automotive industry.

Contents

1 Introduction

1.1. Background Information

HedoN is a high-tech electronic development and production company based in Delft. The company was founded in 1979 as a spin-off from Delft University of Technology. HedoN has been Huisman's technology partner since 1979 for hydraulic motor controls. Huisman constructs large hydraulic plants (such as cranes or deepwater pipelay systems) that are controlled by a Hydraulic Motor Controller (HMC), which is designed by HedoN. This HMC handles input and output signals to and from the plant such that it can be operated in a safe manner.

Hydraulic plants need to be commissioned by very specialized commissioning engineers. Huisman has a small team of those engineers, but is interested in expanding that team. The problem with training commissioning engineers however, is that training can only happen at sea, where the hydraulic plants are. A solution to this problem would be to simulate the behaviour of these hydraulic plants. With a simulation system, commissioning engineers could be trained on land which greatly reduces training costs and in turn enables Huisman to afford more specialized commissioning engineers. Furthermore, a simulation provides a safer environment for commissioning engineers in training to make mistakes.

Another application of the simulation system could be for the engineers designing the HMC at HedoN. Right now the HMC05 has been developed, but software fine-tuning based on results from a simulation could still be very helpful to improve the HMC. For example, the latest HMC supports up to sixteen motors with sixteen encoders, but has never been tested since such a hydraulic plant doesn't exist yet.

An important design consideration concerning the simulation system is that it has been designed with modularity in mind. This will enable HedoN to perfect its simulation system with Huisman and then potentially branch out to other industries. Automotive and aerospace industries for example could also benefit from such simulation systems.

1.2. Simulation System

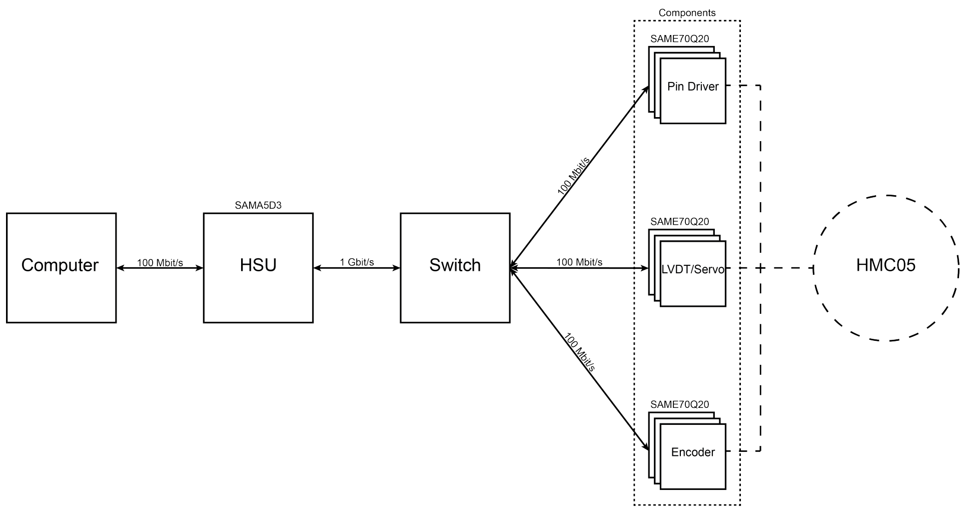


Figure 1.1: System Overview

In Figure 1.1 a schematic overview of the simulation system is shown. On the right hand side of the figure the HMC05, developed by HedoN, is displayed. This hydraulic motor controller would normally be connected to a hydraulic plant in order to control it. That plant has now been replaced by the simulation system.

In order to mimic the behaviour of a hydraulic plant, various electrical components are required to generate an analog signal for the HMC, namely pindrivers, LVDT's, servo's and encoders. Generic pindrivers are able to create a voltage or a current and measure voltages and currents. They are generally used to model sensors and servo's; elements with non-periodic in- and outputs. These components then communicate through Ethernet with a Hydraulic Simulation Unit (HSU). The HSU runs hydraulic simulation code that resembles the dynamic behaviour of the hydraulic plant. Also, there is another Ethernet connection to an external computer/web-server to give relevant information to the operators of this simulation system.

As per assignment for the bachelor graduation project, a group of six students had to be split up in three sub groups. Group 1 has design the pin driver, group 2 has designed the Ethernet communication protocol and group 3 has design the software framework on the HSU and a link to the computer/web-server.

1.3. Problem description

This thesis will focus on the HSU, which runs the simulation and communicates to all the I/O hardware modules (with an embeded microprocessor and Ethernet port). It is important to design the HSU architecture in a robust, modular and flexible way such that it can be used for a wider variety of simulations. The HSU will be connected to the HMC and will simulate the hydraulic plant. This type of simulation is often referred to as a hardware-in-the-loop (HIL) simulation.

Figure 1.2: 120 meter hydraulic crane, built by Huisman and controlled by the HMC

1.4. General Architecture

The simulation system is designed as a Hardware In the Loop simulation (HIL) [10]. HIL simulations are used to test controllers with a simulation of the plant. In normal situations a HMC controls the hydraulics of a crane (usually offshore). An example of such a crane is given in Figure 1.2. Figure 1.3 shows the basic concept of a hardware-in-the-loop simulation. Since the HMC05 controller has a system tick of 1 millisecond (1kHz), the simulation system must also have a system tick of exactly 1 ms (more on this in the programme of requirements, Chapter 2). The setpoint is the place were the hydraulic system has to move the crane. This is an input on the HMC, which decides based on this information how it has to control the plant (hydraulic system).

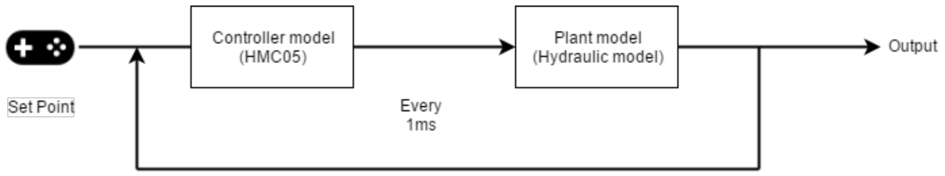


Figure 1.3: Hardware in the loop simulation

1.4.1. Hardware I/O modules

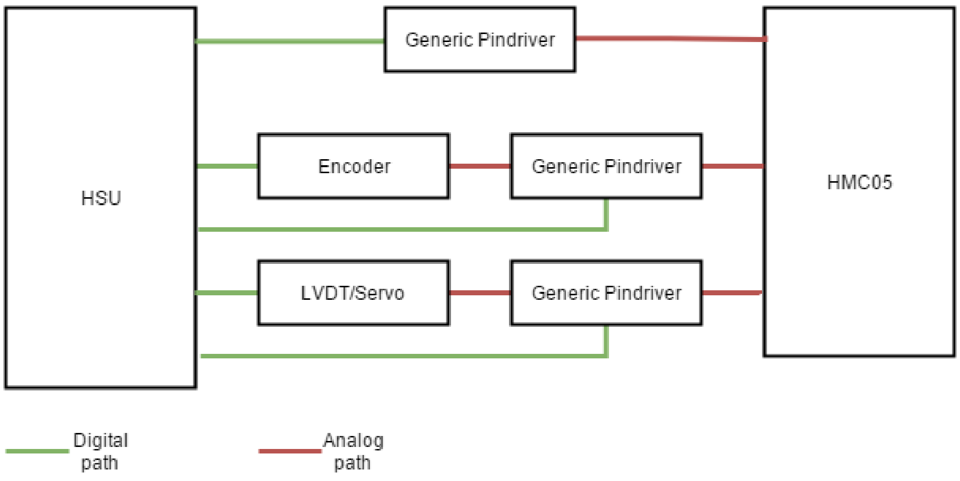


Figure 1.4: Hardware IO units in the system

The Hydraulic Simulation Unit (HSU) is the master of all hardware I/O modules. These hardware modules can generate and/or measure analog signals based on commands from the HSU. To simulate a hydraulic plant, a number of I/O modules types are necessary: Generic pindriver (currents between ± 3.5A or voltage between ± 35V), LVDT/Servo and encoders. Generic pindrivers are either a voltage or current source and can measure both voltage and current while doing so. The generic pindriver can be used to mimic most HMC05 pins, except analog signals that need a sampling frequency higher than 1 ms. Additionally, the generic pindriver has a relay-mode. Instead of being a voltage or current source, it can relay analog signals originating from for example an encoder. This relay mode can be switched on and off remotely from the HSU. This mode is used to simulate a sudden broken cable by essentially placing a switch between an hardware module and the HMC05 input.

All hardware I/O modules must communicate with the HSU via ethernet. A communication protocol for this has been designed. The generic pindriver, encoder LVDT/Servo hardware I/O modules and the communication protocol designs all fall outside the scope of this thesis.

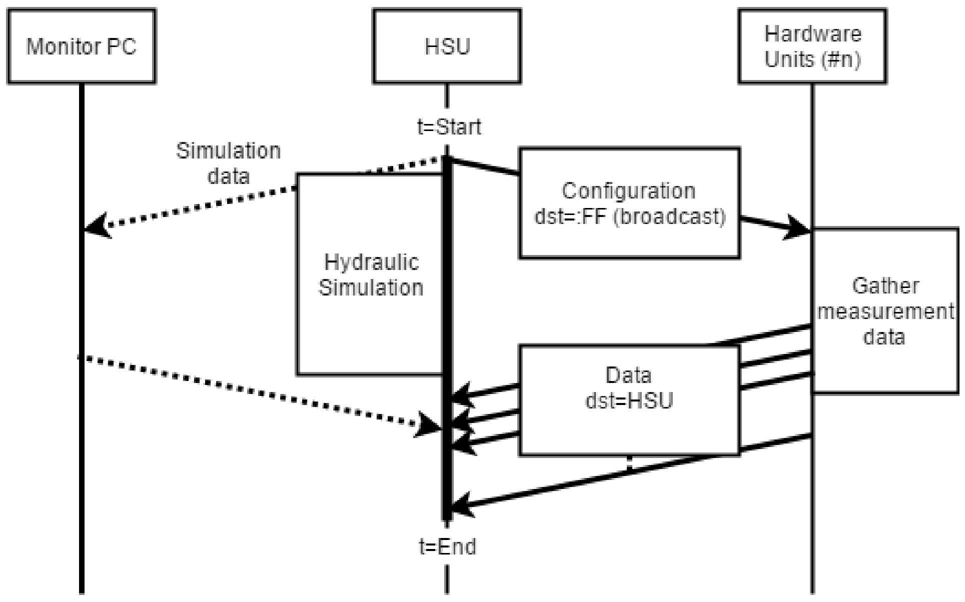


Figure 1.5: One iteration of the simulation system visualized

Figure 1.5 visualizes the 1 ms tick of the testing system. At the start of the iteration, the HSU sends out the same broadcast to all the I/O hardware modules. Each hardware module is configured to read a specific part of the broadcast packet and reply with an ACK or with measurements (sensor vs actuator). This communication is done over low-level Ethernet OSI level 2, to decrease data overhead and computation time. After the HSU has sent out the broadcast, it will run the simulation based on measurements from the previous cycle. Meanwhile all incoming Ethernet packets will be stored in the Ethernet socket buffer. As soon as the simulation is complete, the broadcast packet for the next iteration will be *generated* and the Ethernet receiving buffer will be processed. Optionally, instructions from the monitoring computer are received and executed here. As soon as the 1 millisecond is over, the previously generated broadcast packet (with new payload) is sent out again and the cycle repeats.

1.5. Overview

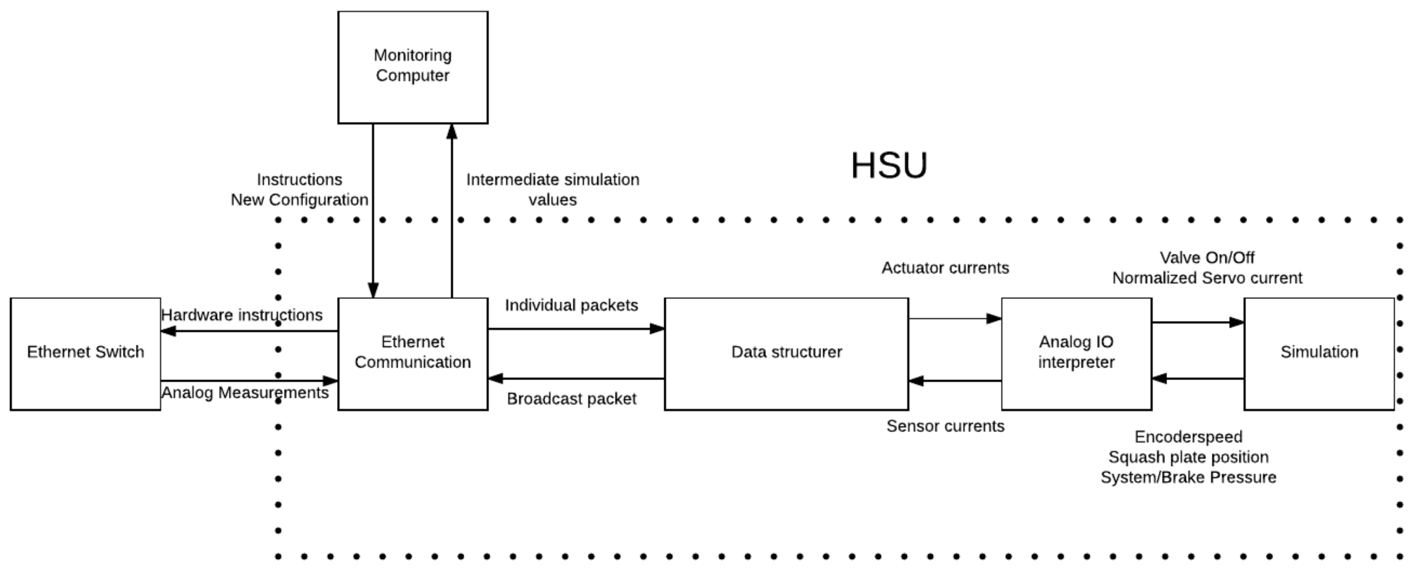


Figure 1.6: Overview of submodules and the interaction between

The proposed design in this thesis can be divided into a number of sub-components. The data interaction between each of the modules can be found in Figure 1.6. First the HSU hardware and operating system will be explained in Chapter 3. The Ethernet module will be explained in Chapter 4, the data structurer and analog IO interpreter will be discussed together in Chapter 5, the simulation system will be discussed in Chapter 6. In Chapter 7 the monitoring computer will be explained. Finally in Chapter 8 all modules will be integrated and the initialization, configuration and interaction between the components will be discussed. The thesis will end with the test results (Chapter 9), the discussion (Chapter 10) and the conclusion (Chapter 11).

2 Program of Requirements

The hydraulic simulation system will be connected to the HMC05. This controller has numerous analog I/O that are normally connected to hydraulic/electric sensors and actuators. The hydraulic plant simulation must simulate the behaviour of these components such that the HMC05 control feedback loop works properly. The requirements are based on HedoN problem description documents [18] and [14] and various conversations with HedoN engineers.

2.1. Function Requirements

The functional requirements determine what the design must do and what functionalities it must have based on the problem description.

Requirement	Description
1	The simulation system must interpret all analog outputs of the HMC
2	The simulation system must provide analog inputs to the HMC
3	The simulation system must be able to simulate a broken cable or component
4	The HSU must simulate the hydraulic plant based on the same models used to develop the HMC
5	A monitor computer must visualize, monitor and store simulation results
6	A monitor computer must be able to send instructions to the hardware modules
7	The simulation system must work without a computer
8	The simulation system configuration must be stored on non-volatile memory
9	The simulation system must have a tick of 1ms (1kHz)

2.2. System requirements

System requirements are non-functional requirements that define how the system is utilized. In this project, it defines the roles of HedoN engineers, Huisman Commissioning engineers and students and their use cases.

Requirement	Description
10	A Commissioning Engineer must be able to fully configure the hydraulic plant
11	A Commissioning Engineer must be able to instruct the system to break components/cables
12	A HedoN Engineer must be able to expand the hydraulic simulation
13	A Commissioning Engineer must be able to extract and save raw simulation data from the HSU

2.3. General Conditions

The general conditions are determined by the framework in which the project is carried out. There are guidelines and assumptions from HedoN and requirements laid upon by the other thesis subgroups.

Requirement	Description
14	The HSU will communicate over Ethernet OSI level 2 with hardware components
15	The HSU has a master role in the communication with the hardware
16	The hardware components must be identified by their mac address
17	The communication protocol and instruction list determined by subgroup 2 must be implemented on the HSU
18	An Atmel SAM A5 cortex MPU must be used to implement the HSU

3 HSU environment

The HSU is implemented on a SAMA5D3 [6] (general Requirement 18). This is a Cortex M5 based microprocessor prototyping and evaluation platform which is one the fastest developer boards by Atmel. HedoN provided a Linux distribution stripped of most peripherals except two ethernet ports.

3.1. Microprocessor

The SAMA5D3 has a fast processor which can run heavy simulations. It also has an Floating Point Acceleration Unit. The truncating noise on single precision floating point is extremely low [8] and has been implemented in all sub components of the simulation system.

The SAMA5D3 development board has two distinct Ethernet ports (a 100MB/s and 1Gb/s port). Both ports have been used in the final design. The gigabit port is used to communicate to all the hardware I/O modules, the other is used to communicate with the monitoring computer. It is essential to isolate the monitoring computer from the hardware I/O modules, as the latter have arbitrary mac addresses and may never be connected to the internet, whereas the monitoring computer must not have such constraints laid upon.

3.2. Software framework

For the simulation system to work properly according to Requirement 9, a system tick of 1 ms must be achieved. In order to be real time a choice in developer environment had to be made. It is important to understand that realtimeness is a relative factor in microprocessors as microprocessor are discrete in timing. The SAMA5D3 can be run bare-metal but there is no support for any peripherals on the developer board. To overcome this, the HSU will run on a stripped version of Linux. The Linux build has been provided by HedoN, the possibilities to switch to a real-time linux OS such as RTOI have been considered but the lack of a compiled kernel specifically for the SAMA5D3 made this unfeasible in the time frame of the project. The stripped Linux kernel is expected to be still close enough to real time as only the most basic functions on the operating system are running. However this needs to be verified by tests.

3.3. Programming language

All custom software running on the HSU has been programmed in C++. The GNU compiler has been used to compile to the SAMA5D3 microchip running linux. The choice for C++ is mainly based on the need for low level memory access (necessary for the data structurer, Chapter 5) and the ability to switch back to bare-metal at very little cost at any time. Furthermore, C and C++ are part of the curriculum of the bachelor Electrical Engineering at the University of Delft, which made it a comfortable choice with no steep learning curve.

3.4. Threads

The simulation will run on a Linux kernel which enables to possibility for multiple threads. Figure 1.5 shows that during the 1 ms tick the listening and simulation are independent processes. This makes it possible to run these processes asynchronous parallel, in which case the listener thread would always listen to incoming packets and a separate thread would only run the simulation and send the broadcast package. However, this is not implemented as this demands shared memory between two threads which complexifies the software framework and possibly causes great synchronization problems. As the SAMA5D3 is a single core processor, the listener thread and simulation thread would be constantly competing for computation time. If however a consecutive version of the (hydraulic) simulation system would be implemented on a multi-core processor, the use of multiple threads could greatly increase performance.

4 Ethernet Communication

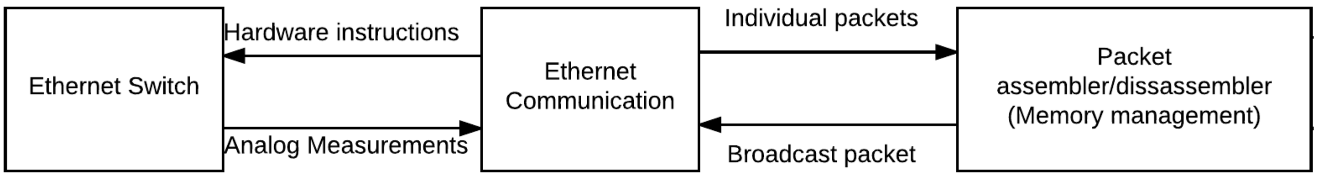


Figure 4.1: Overview of data in and outputs of the ethernet module

The simulation engine communicates every iteration with both the PC and the hardware units (Figure 4.1. These packets are sent during the send packet stage in the 1 millisecond loop. There also exists two receive phases. One receives packets from the hardware units and one which can receive instructions from the computer. Figure 4.2 shows the HSU interfaces. The connection to hardware units (the 1Gbit port) is required, otherwise the HSU can't function. However the 100Mbit link to the computer is in both ways optional as the system is required to work without an PC (Requirement 7).

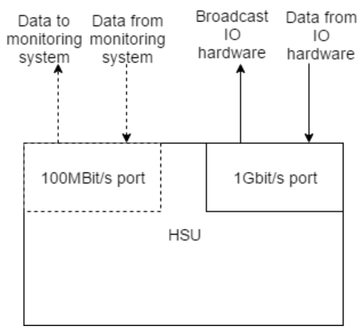


Figure 4.2: Ethernet interfaces of the HSU

4.1. Responsibilities

The responsibilities of the ethernet communication system can be divided in two parts. The basic responsibilities of the HSU to handle the hardware IO units.

- Receive 1 big packet from the packet assembler (Chapter 5) and must broadcast this to all hardware modules

- Read Ethernet packets originating from hardware IO modules out of the Ethernet buffer and send this to the packet assembler

When a monitoring system is connected, the HSU is also responsible for this connection. In this situation the Ethernet communication is also responsible for the following functions.

- Receive the 1 big packet with the information from the packet assembler (Chapter 5) and send this to the monitoring system
- Read Ethernet packets from the monitoring system and send them to the assembler, these are used to overrule the simulation (eg. simulate a broken cable connected to the encoder)
- Load a custom setup for a simulation on the HSU, this is loaded during the initialization of the simulation

4.2. Low-Level ethernet

The Ethernet communication throughout the simulation system is done on OSI-layer 2. This design decision is defended by the accompanying thesis of Jorden Kerkhof and Sam de Jong [4]. On this level, the Ethernet packets routing is based on the mac addresses (rather than a local IP address). The use of this low level Ethernet protocol reduces the computation time (as no checksums need to be computed) and overall decreases the overhead (smaller header, same payload).

4.3. Ethernet initialization

Sockets for sending and receiving are created during the initialization phase. Sockets are the connection between programs, in this system they route information between the HSU and the Ethernet port. Creating sockets is a time consuming process. From test results from thesis [4] it is know that creating a socket each time cost around 4 times as much time as pre-initializing the sockets. Therefor the sockets are only created during the initialization phase, after which the same sockets are used in the system loop of 1 ms.

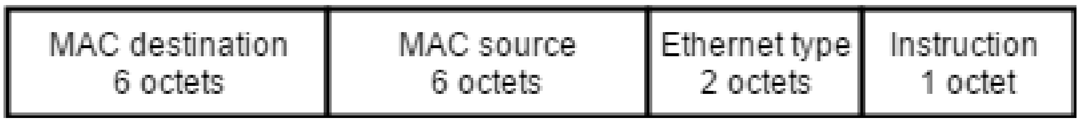
This module uses Linux kernel based libraries [9] to create two or four sockets (one or two to receive and one or two to send). These sockets are configured to be RAW, as this is a general condition (Requirement 14). For hardware IO communication this module must first be initialized (create sockets, pre-compute the broadcast header), after which it can rapidly send out broadcast packets and relay received packages to the packet (diss)assembler. For the monitoring system a RAW ethernet socket is configured as well, but instead of being a broadcast packet, it is rather a unicast to the mac address of the monitoring computer.

4.4. Sending data

The sent data function is implemented to have the payload first byte, the length and the destination mac address as inputs. To optimize the sending of the broadcast, this function has a precomputed broadcast header that it can use if it is instructed to send out a broadcast. Additionally, the data payload is communicated to the function by reference, to optimize for execution time, this data is not physically moved around, but only passed to the socket. The opening of RAW sockets and the creation of the header is based on a code snipped created by Austin Marton [3].

4.5. Receiving data

The receiving data function checks whether there are packets in the Ethernet buffer. The Ethernet controller is a seperate hardware controller on the SAMA5D3 chip. When the microprocessor receives a packet, it is first written to a register on the Cortex M5 processor. This is a FIFO shift register. The receive function will read the oldest packet in this register. If no packets are present in the FIFO register, the function returns -1. This code is also also based on a code snippet created by Austin Marton, found in Github [2].



MAC destination 6 octets	MAC source 6 octets	Ethernet type 2 octets	Instruction 1 octet

Figure 4.3: Ethernet header

The header of a level 2 packet looks like Figure 4.3. The first part is reserved for the destination MAC-adres, in the case of a broadcast packet this is FF:FF:FF:FF:FF:FF. The second block is fixed for the source mac address. The next block is defined as the Ethernet type, this is fixed to "0x88B5", the Ethernet type reserved for 'experiments'. The last byte in the header is not part of the IEEE standard, but is rather added by the communication designers of the simulation system [4]. This byte is the instruction byte. This is use to quickly determin the purpose/instruction of any packet on the network at relative low computation cost. More information about the communication protocol between the HSU and all the hardware IO modules can be found in the accompanying thesis [4]. The initialization of the Ethernet module will also be discussed seperately in Section 5.4.2.

Note that the HSU is connected to on or two private networks: One with all the IO hardware modules and optionally one with the monitoring computer. Neither of these private networks are connected to the world wide web, since the mac addresses used are not registered at the IEEE-SA.

4.6. Limitations

The Linux OS automatically attaches the header when sending packets, this means that the source mac address cannot be spoofed. This limitation will become relevant when testing the initialization of the data structurer.

5
IO interpreter and data structure

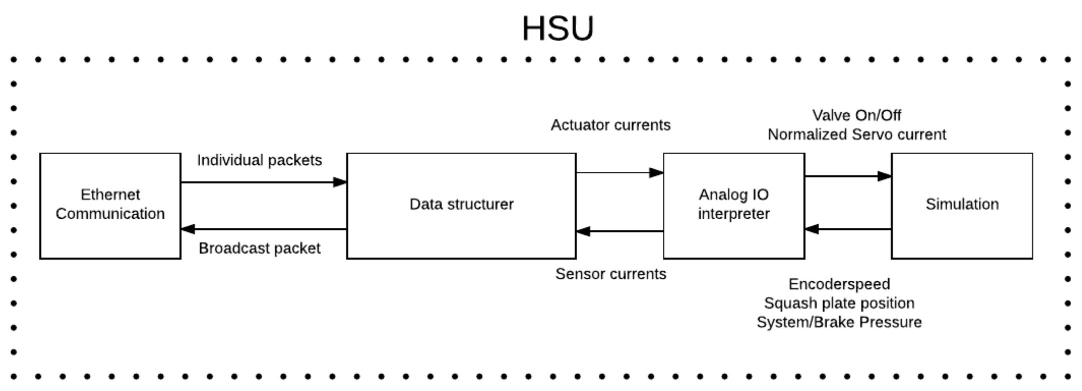


Figure 5.1: Overview of IO interpreter and data structurer

In overview Figure 5.1 it is shown that the raw data from the Ethernet module must be translated to simulation friendly units (eg. a measured voltage is converted to valve on/off). This is done by the IO interpreter modules. For every in and output *type* of the HMC a different IO interpreter module exists (ie. encoder, pressure sensor etc). In this chapter, first all HMC I/O will be discussed, from which all IO interpeters will emerge. After which, the data structuring module will be discussed. This module is the master of all interpreters and allocates memory for each IO interpeter module. This module optimizes the way the broadcast ethernet packet is built and unwraps and sorts the incoming ethernet packets. The module has been designed in such a way that the simulation system is extremely modular and flexible. This will be discussed in the last section of this chapter.

5.1. HMC In and Outputs

As discussed previously, the HMC has many in and outputs. In this section all different in and outputs will be discussed. Each I/O will have a (custom) hardware translating module. The generic pindrive can simulate a number of I/O pins.

Warning: The use of 'input' and 'output' can easily be confused in the design of a HIL simulation. Throughout this thesis all I/O will be considered from the simulation system point of view.
HSU Outputs (see Requirement 2):

- Encoder: Generates square wave pulses based on angular displacement. A hardware module has been developed that can simulate quadrature encoders (this falls outside of the scope of this thesis). The simulation must provide the **angular velocity** and the encoder will receive the **instantaneous pulse frequency** of the square waves and phase shift (direction).

- LVDT: The LVDT is a Linear variable differential transformer that measures the swash plate angle (explained in Chapter 6). The simulation must provide the **nominal swash plate position** $\in [-1, 1]$ (see Chapter 6). This will be directly relayed to the hardware module.

- Pressure sensor: A pressure sensor behaves likes a variable current source. The sensor current output has a linear relationship to pressure. This component can be implemented with a generic pin driver ([5]) in current sending mode. The simulation must provide the **pressure (bars)** and the pindriver will receive a **0 to 25mA DC current**.

- Torque balance sensor: This is a binary sensor that can detect when the motor torque is balanced with the load torque (more on this component in Section 6.2.8). This input can be implemented with a generic pin drive in current sending mode. The simulation must read a **true/false state**, the pindriver will receive a ± **20mA DC current level** .

HSU inputs (see Requirement 1):

- Servo: The HMC provides a modulated **analog current signal** (±125mA DC component). This must be translated to a **nominal AC current** $\in [-1, 1)$ for the simulation.

- Binary valve: The HMC provides either **0 or 1.2A**. This must be translated to the state of the valve **open or close**, for the simulation. This is implemented with a pindriver.

5.2. The IO interpreter module

From the list of I/O modules the HMC05 has, one can derive that a total of four different hardware modules are necessary to generate and measure all analog signals:

1. Pindriver for pressure sensors, various hydraulic valves and the torque balance sensor
2. Servo
3. LVDT
4. Encoder

The IO interpreter modules have been implemented as C++ classes. Inheritance has been used to systematically define the distinction between for example a generic pindriver, a pressure sensor and an encoder. The overview of the inheritance can be seen in Figure 5.2. The figure shows that the parent class must have a (unique) mac address defined and an inbox and outbox location (pointer) and size, more on this in the next Section. Each IO interpreter must also have an offset that determines what part of the broadcast to read.

Childs of the parent class can define for example the payload data structure and consequently the size of the inbox and outbox. As an example it is known that the pindriver sends out a mode, two floats (voltage and current) and the temperature. More on this in the example of this chapter 5.4.3. The pindriver then can have childs that specify the purpose of the generic pindriver. As an example it can be a hydraulic actuator (defining that the pindriver must measure currents, determining the mode and range), or the pindriver can be for example a pressure sensor (defining the pindriver as a variable current source).

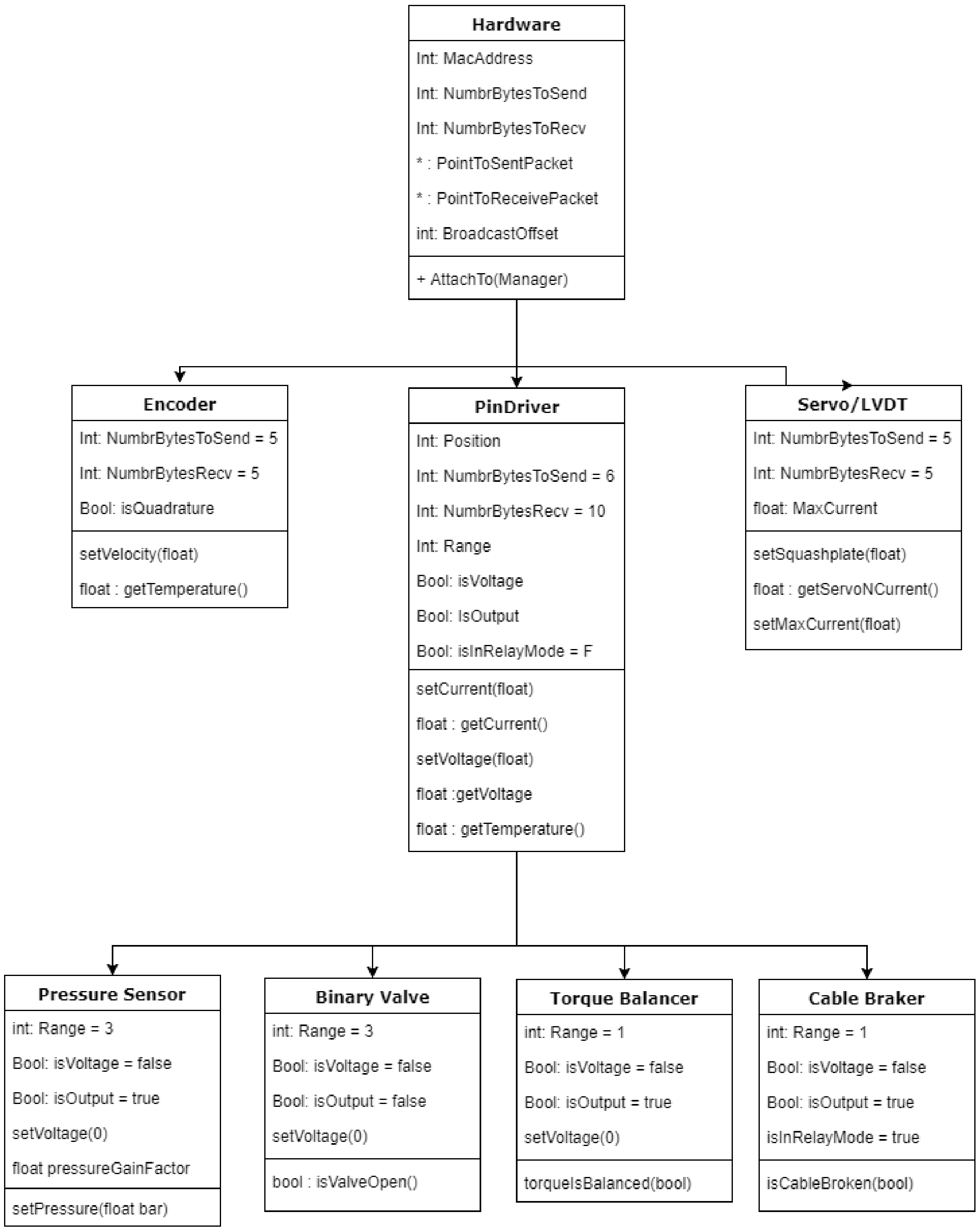


Figure 5.2: Hardware class inheritance visualized

5.3. The data structurer

In Figure 5.3 one can see that each IO interpreter module has an inbox and an outbox. The data structuring of the inbox and outbox is defined by the type of interpreter module (example ahead in Figure 5.6). The location of the inbox and outbox is **not** determined by the IO interpreter itself, but is rather *centrally* assigned by the data structurer. Each IO interpreter module has two pointer that points to the in and outbox.

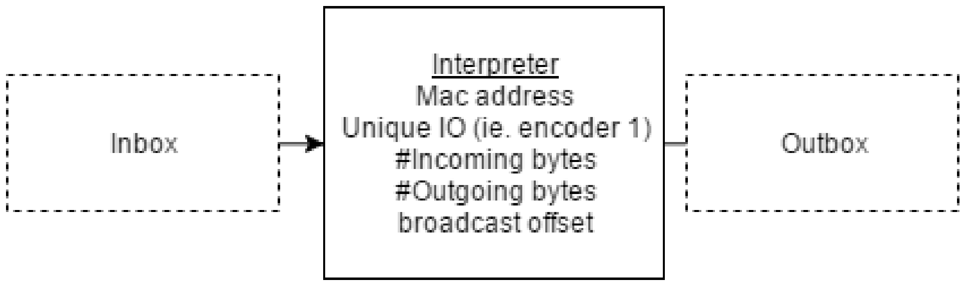


Figure 5.3: Visualization of the I/O interpreter

5.4. Interaction IO interpreters and data structurer

To best illustrate the interaction between the IO interpreter and the data structurer and its functionality, first the different responsibilities will be given, after which the initialization of both modules will be discussed and lastly a simple example will be given.

5.4.1. responsibilities

The responsibilities for the **data structurer** are:

- Create a format for the broadcast package based on all hardware interpreters that are attached
- Give pointers to the inbox and outbox of each IO interpreter module.
- Tell each IO interpreter module what offset to use when reading the broadcast package. This information must be shared with the physical hardware component by ethernet.
- Handle a received package by storing the relevant content in the correct inbox (this is determined by the unique mac address each IO interpreter has).

The responsibilities of the **IO interpreter** modules are:

- Has a separate module for each hardware component (I/O) that must be simulated (ie. two pressure sensors are two different objects, each has unique mac address).
- Each component knows the communication packet structure (both incoming and outgoing, example Figure 5.6). Furthermore, it knows how to translate raw values (such as a float representing a current), to physically relevant value like a pressure in bar.
- Receive a pointer from the data structurer for both the inbox and outbox. This is a unique location for each module.
- Must also store the physically relevant values (ie. pressure in bars), to be able to communicate those to the external computer.
- Must attach itself to the data structurer (ie. responsibility lies in the IO interpreter to make itself known the the data structurer).

5.4.2. Initialization

To illustrate the interaction between the IO interpreter modules and the data structurer the initialization phase will be explained. The initialization phase consists of the following steps.
Note: Before the initialization all I/O hardware modules connected to the switch may receive broadcasts but have no idea as to what part of the broadcast contains their specific information. This is by design. More on this in the thesis of Sam & Jorden [4]

1. Attach all IO interpreters to the data structurer
2. The data structurer determines the size of the broadcast package and all data to be received, and reserves this memory (malloc). Consequentially it also computed the broadcast read offset for each component.

3. The data structurer will update the pointer to the inbox and outbox of all IO interpreters.

4. The data structurer now has a list of all IO interpreters and their unique mac addresses. The data structurer will send a personal **ethernet** package to all the mac addresses and demand an acknowledgement.

5. If the hardware responds with an ACK, the packet assembler sends out an integer containing the offset necessary to read the correct part of the broadcast.

6. The handshake is finalized as the hardware component repeats (echos) the offset it will utilize to read instructions from the HSU.

5.4.3. Example

To best illustrate the functionality of the IO interpreter modules and the data structurer, an example will be given.
When configuring the simulation system, a number of arbitrary decisions must be specified, namely a list of all hardware modules, their mac address, module type and purpose must be given.

Mac	Type	Purpose
1	Pressure Sensor	Boost pressure
2	Encoder	Attached to motor 1
3	LVDT	Attached to motor 1
4	Servo	Attached to motor 1
5	Pressure sensor	Brake Pressure

Table 5.1: Example of configuration list specified by user (eg. Huisman Commissioning Teacher)

Note that the type pressure sensor already implies the use of a pindriver (see Figure 5.2).
All design decisions are based around the fact that the list provided in Table 5.1 is the only list that is hard coded. The design discussed is extremely flexible and modular and is purely dependant by above list.

Based on the list provided in Table 5.1 the data structurer can now reserve the memory receive block and the memory send block. These are illustrated in Figure 5.4 and 5.5 .
Note that the memory send block starts with the byte *0x04*. This is the instruction byte for the broadcast package (more on this in the other thesis [4]). Since the ethernet module expects a fully constructed payload (including the instruction) it is most efficient to add it here, to avoid unnecessary moving of memory later.

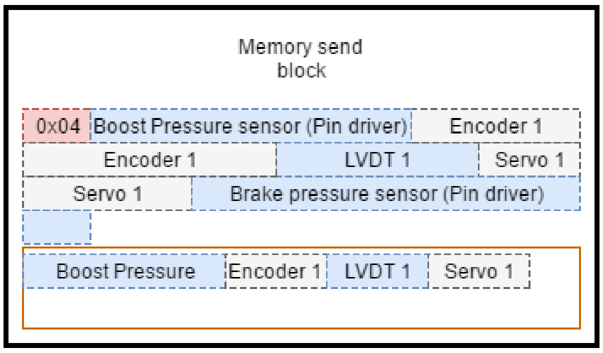


Figure 5.4: Send memory block. Not to scale

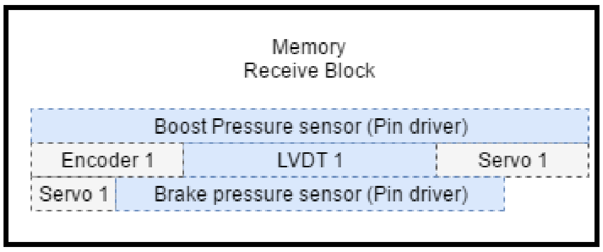

Memory
Receive Block
Boost Pressure sensor (Pin driver)
Encoder 1
LVDT 1
Servo 1
Servo 1
Brake pressure sensor (Pin driver)

Figure 5.5: Receive memory block. Not to scale

This example will zoom into the *Boost pressure sensor (Pin driver)*. The IO interpreter inherently knows the structure of the inbox and outbox. This is determined by the father child of the pressure sensor: The pindriver (see Figure 5.2). The data structure is visualized in Figure 5.6. The choice of structure of the pindriver falls outside of the scope of this thesis.

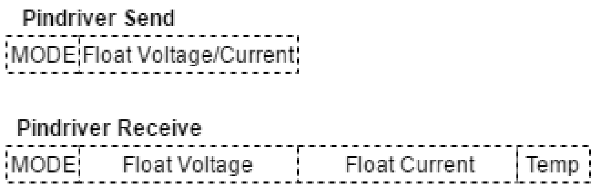


Figure 5.6: Above: Data structure that the HSU must send to pindriver. Below: Data sourcing from pindriver

In this example, the data structurer would send out a total of 5 ACK requests to all hardware modules (on their mac address). After which it would communicate to each hardware module their offset. This is the number of bytes it must ignore starting from the *0x04* byte.

Mac	Type	Purpose	Offset
1	Pressure Sensor	Boost pressure	0
2	Encoder	Attached to motor 1	6
3	LVDT	Attached to motor 1	16
4	Servo	Attached to motor 1	25
5	Pressure sensor	Brake Pressure	31

Table 5.2: The data structurer determines the send block and specifies the offset for each component

Since each IO interpreter now knows the location of its inbox and outbox, it can now translate the incoming analog measurements to values relevant for the simulation. In this case the boost pressure sensor (reminder: this is a variable current source, Section 5.1) would instruct the pindriver to force a voltage of zero. The incoming current float (Figure 5.6) would then be translated to a pressure in bars. This value can then be requested from the simulation.

6 Simulation

6.1. introduction

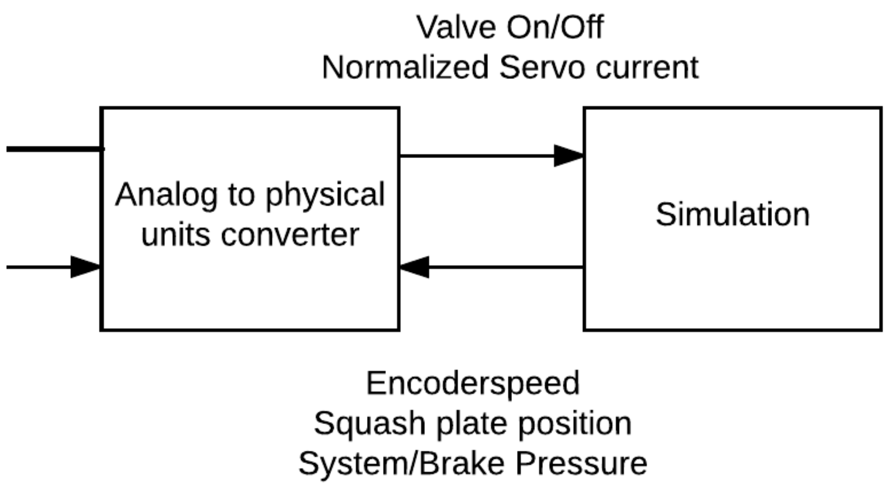


Figure 6.1: Overview of data I/O of simulation module

The simulation module is the next module of the HSU to be discussed in this thesis. The simulation module is responsible to simulate all the dynamic behaviour of the hydraulic plant. Figure 6.1 shows that the module receives information from the HMC05 through the hardware interpreter. The information going into the simulation module is very rich in context (ie. The HMC05 wants to turn on the safety valve, instead of pin2 = 20 volts). This information is refreshed every 1 ms. The simulation must compute all new values for each observation point that the HMC has (various pressure sensors, rotary encoders etc). The simulation module must push these new values back to the hardware interpreter and from there it will be wrapped into an Ethernet packet and broadcasted to all hardware IO modules connected to the switch. The time between each broadcast must be exactly 1ms. The simulation must therefore compute for $t = t_0 + 1ms$ in less than 1ms computational time.

In this chapter, first the simulation method will be explained, after which a short introduction into hydraulics is given (Section 6.1.3), specifically about the hydraulic motor type that is controlled by the HMC (Section 6.1.4). A number of hydraulic and mechanical components are necessary for the motor to operate. All these components will be converted into simulation blocks. By connecting and configuring these blocks a wide variety of hydraulic setups can be simulated. Finally, the flexibility and configurability of the simulation engine will be discussed.

6.1.1. Simulation method

Since the simulation must step forward exactly 1 ms each iteration, a numerical iterative simulation is chosen. The input values of each iteration may differ (based on the HMC outputs) and each iteration all sensor values must be recalculated and pushed to the hardware IO modules. The simulation must numerically solve integrals explicitly. For this the Euler method can be used. The Euler method is a first-order method, which means that the local error (error per step) is proportional to the square of the step size, and the global error (error at a given time) is proportional to the step size [19].

Step size

As can be seen in Figure 6.2, each millisecond the system must present values to the controller. However, this does not restrict the step size to 1 ms, but rather gives an upper boundary. To increase stability or reduce errors it is possible to iterate multiple times with a lower step size. It is essential however, to step forward exactly 1ms each system tick.

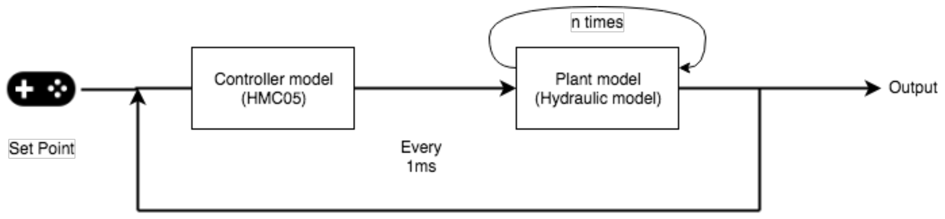


Figure 6.2: Flowchart of Hardware-In-The-Loop, illustrating a multi-iterative simulation engine

Stability

The explicit Eulers method can be numerically unstable (also referred to as the *standard* or *forward* Eulers method). Stability can be restored by either reducing the step size, or by using the implicit Eulers method (the *backwards* Eulers method). An implicit method evaluates the end point of the step, rather than the starting point. Such an implicit method is more costly to implement [19]. The general requirement of 1ms gives an upper boundary of 1ms as a step size for the simulation module. With such a small step size, the backward Eulers method is found to be stable for all simulation blocks.

Noise

With a step size of 1^{-3}s both the local and global errors are negligible. Furthermore, the simulation has been implemented with floats which introduces a truncating error that is negligible as well [8].

6.1.2. Software

The simulation has been written in C++. This was chosen because C++ will always be backwards compatible with bare-metal, should at any point the decision for Linux be withdrawn. Additionally C++ allows for object oriented programming, this has been used to implement the simulation. All hydraulic and mechanical components have been implemented as software objects.

6.1.3. Secondary hydraulics

Conventional (primary) hydraulic motor circuits are designed with a *variable* hydraulic pump and a *fixed* hydraulic motor. The motor power output is therefore proportional to the hydraulic pressure generated by the variable pump. Since the hydraulic pump has a high moment of inertia, this type of system design has inherently low controlled acceleration.

Secondary hydraulics is a closed hydraulic circuit where the hydraulic motor has a *variable* volume displacement. A pressure regulated hydraulic pump provides a *constant system pressure*. Thus, by varying the volume displacement of the hydraulic motor, the output power (or torque) can be controlled. To control the variable hydraulic motor, a proportional valve (servo) and a positioning cylinder are needed. Secondary hydraulics has a low response time and can reach very high controlled acceleration. Another advantage is that a single pump system can control multiple independant loads (motors). The disadvantages of secondary hydraulics is the complexity of the design. Furthermore, safety valves and backup components are necessary to guarantee a *normally safe* design. A more in depth explanation about secondary hydraulics can be found in the first chapter of *Secondary Controlled Swing Drive* by Karl Pettersson [13].

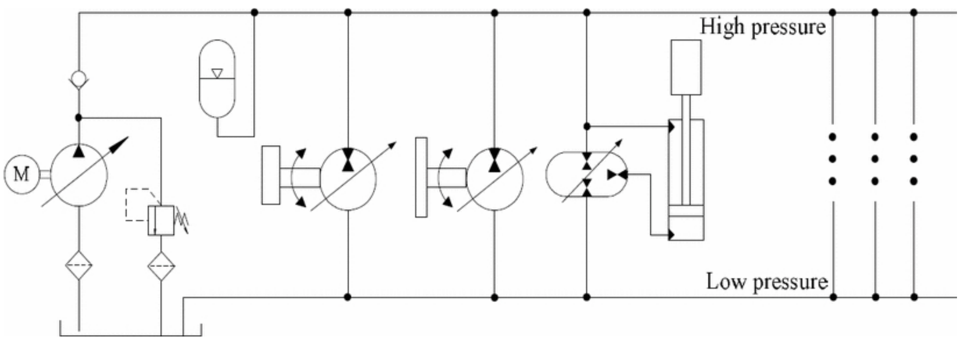


Figure 6.3: Secondary hydraulic circuit with a pressure regulated pump on the left and various loads on the right [16]

In Figure 6.3 on the left the pressure regulated hydraulic pump is shown. Multiple loads can be connected on the high and low pressure rails. The low pressure (also known as boost pressure) pumps are not shown in this image. The system pressure is equal to the difference between the high and boost pressure rails and is typically referred to as ΔP.

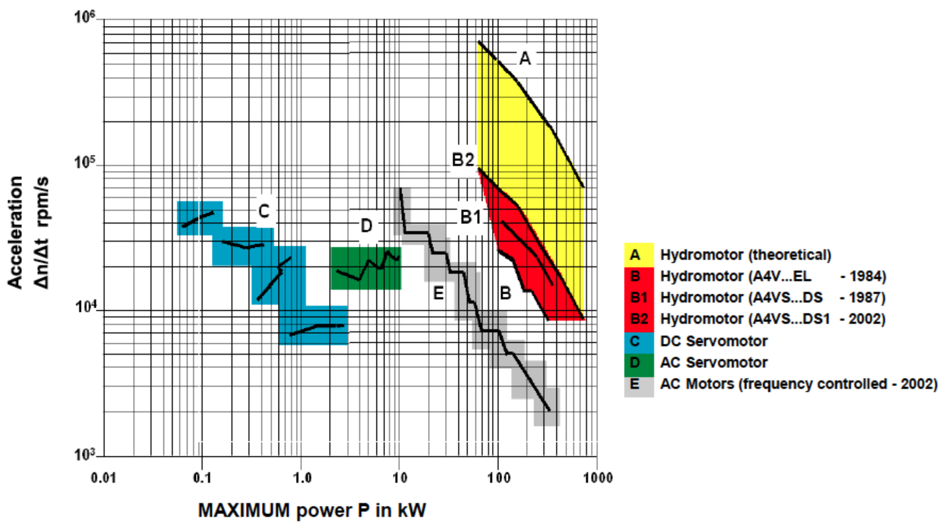


Figure 6.4: Acceleration versus maximum kinetic power. Comparison of different types of motors. In red motors controllable by HMC. [1]

Figure 6.4 shows that hydraulic motors have high **controllable** acceleration and high output power. Furthermore, the physical footprint of such motors is much smaller compared to electric motors [1].

6.1.4. Hydraulic Axial Piston Motor

The HMC of HedoN is designed to control a hydraulic axial piston motors. These motors have an adjustable swashplate position that controls the volume displacement and consequently the torque/output power. When these motors are configured in a secondary hydraulic configuration (constant pressure system), these hydraulic motors can work both as pump and motor. It can provide positive and negative torque at any speed. This is often referred to as a four quadrant system [13].

In figure 6.5 one can see the a cross section of the axial piston pump design. The pistons rotate with the outer shaft. The pistons exert force on the swashplate. The swashplate is an incredibly flat mirror-like slippery surface on which the pistons slide. The swashplate **tilt angle** determines the amount of torque (and direction) and is independent of the current rotational velocity nor direction of the shaft.

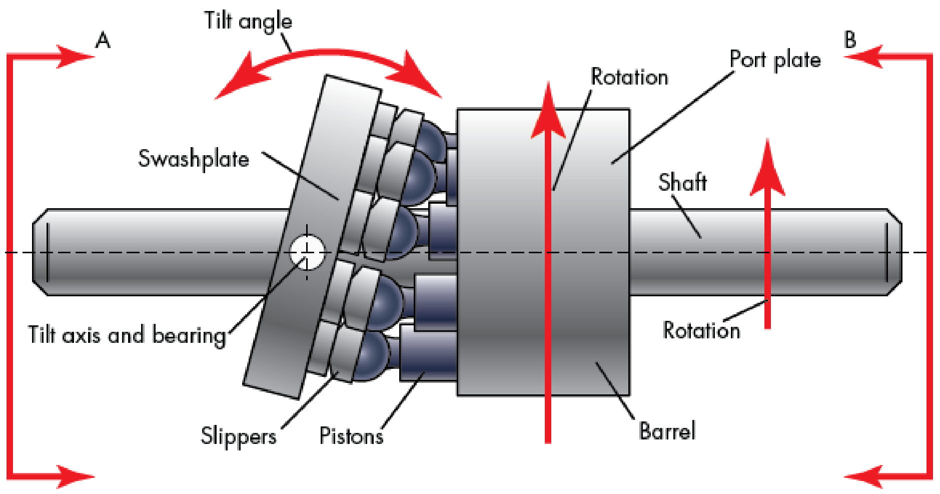


Figure 6.5: Hydraulic axial piston pump cross section [7].

6.2. Complete hydraulic circuit

For the axial piston motor to work safely a number of components are necessary. Figure 6.6 shows a single motor hydraulic circuit. A list of the components will be given, after which each component will be explained in detail, the physical behaviour will be modelled and the resulting simulation block will be derived. At the end of this chapter the simulation blocks will be combined to form a number of different winch based mechanical hydraulic systems.

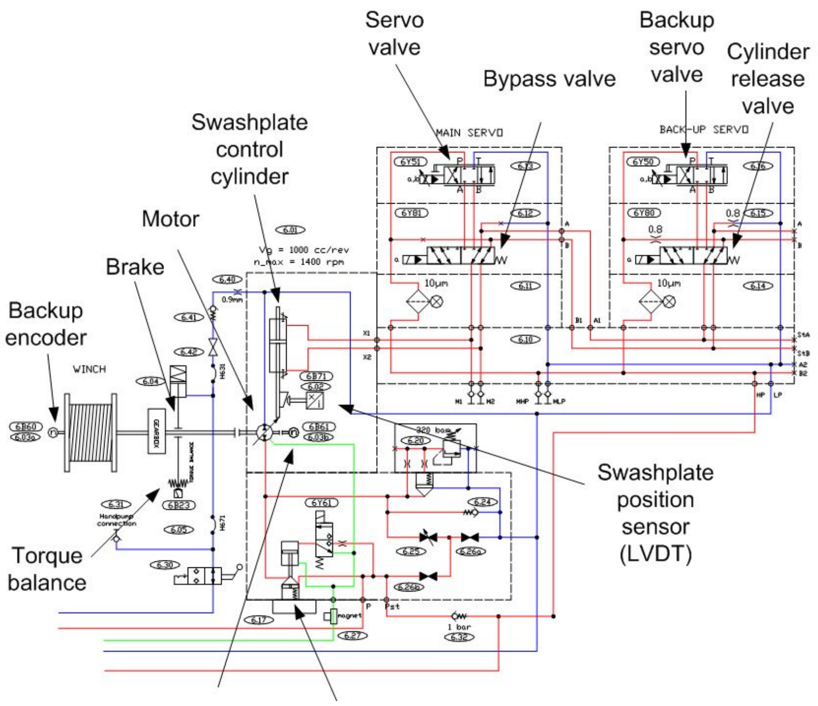


Figure 6.6: Single motor hydraulic scheme [14]

1. Hydraulic Pump

2. Accumulator
3. Axial Piston Motor
    (a) 4-way electro-hydraulic servo
    (b) Positioning cylinder
    (c) swashplate angle sensor (LVDT)
4. Logic valve
5. Bypass servo valve
6. Rotary Encoder
7. Hydraulic-mechanical brake
    (a) Pressure sensor
8. Gearbox
9. Winch
10. Torque balancer

6.2.1. Motor

As explained in Section 6.1.4 the hydraulic motor controlled by the HMC05 is an axial pistion motor. The only manufacturer of these components is the German company Rexroth (Bosch Group).

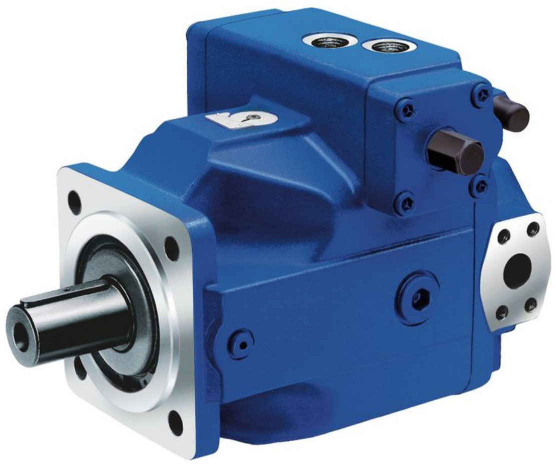


Figure 6.7: The Rexroth AV4SO variable displacement motor

Model

The AV4SO is a typical piston unit. It can be used both as a motor or as a pump. When operating as a motor the output torque and oil displacement are: [15]

$$Q_v = \frac{V_g * n}{1000 * \rho_v} \tag{6.1}$$

$$T = \frac{V_g * \Delta P * \rho_{mh}}{20 * \pi} \tag{6.2}$$

where

$V_g =$ $V_{max} * x =$ Geometric displacement $[cm^3]$

$\Delta p =$ Pressure difference [bar]

n = rotary speed $(min^{-1}]$

$\rho_v =$ Volumetric efficiency

$\rho_{mh} =$ Mechanical-hydraulic efficiency

However Equation 6.1 and 6.2 can be simplified by applying the boundary conditions that $\rho_v \approx \rho_{mg} \approx 1$ and $V_g = V_{gmax} * x$

Implementation

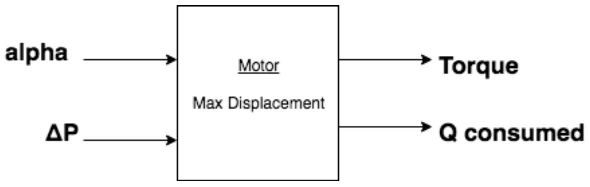


Figure 6.8: Block diagram of motor simulation object

Since the normalized position of the cylinder (alpha) determines the geometric displacement, the simulation block has this as input. It outputs the instantaneous torque and the oil consumed.
The resulting equations become:

$$T(x, \Delta P) = \frac{x * V_{gmax} * \Delta P}{20 * \pi} \tag{6.3}$$

$$Q_v = \frac{x * V_{gmax} * n}{1000*} \tag{6.4}$$

where

x = the swashplate normalized angle x ∈ [-1,1]

$V_{gmax} =$ Maximum displacement $[cm^3]$

Δ P = System pressure [bar] (in practice depends on logic valve state)

6.2.2. 4-way electro-hydraulic servo

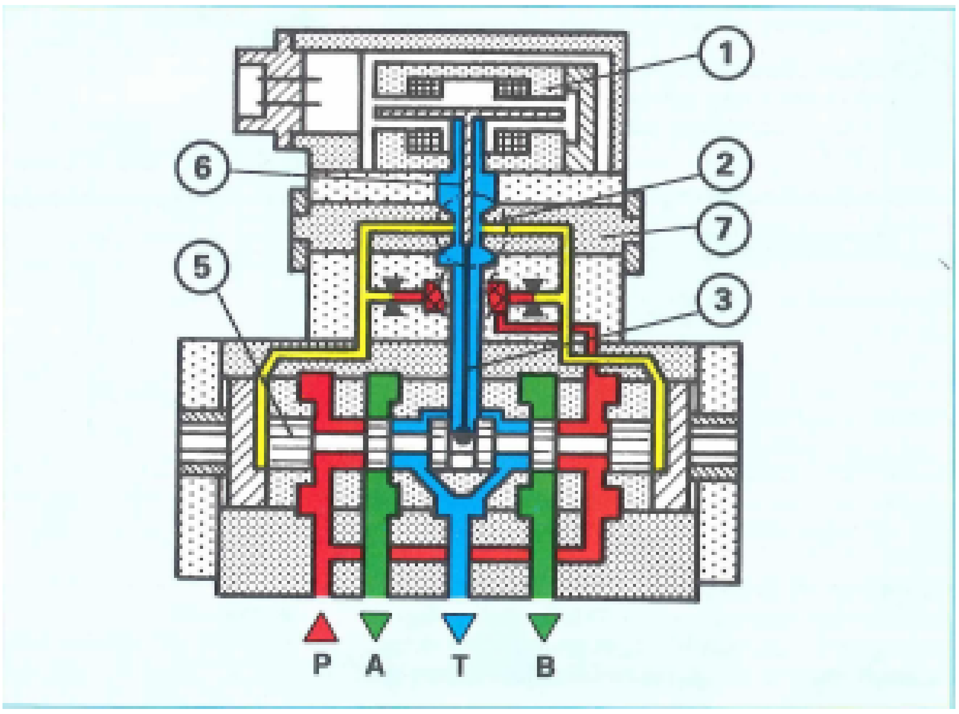


Figure 6.9: schematic overview of 4-way electro-hydraulic servo. source:[1]

An electro-hydraulic servo converts an electric current into a flow. The input current goes through a coil which displaces a proportional valve, which controls the output oil flow. Ideally the relationship between current and oil flow is a linearly constant, however due to the inductive nature of the electric part, the response is frequency dependant [11]. There are two oil tubes as output (referred to as A and B) where the oilflow of $A = -B$.

Model

According to [17] an electro-hydraulic servo can be modeled with a fifth order model. However, only the first natural frequency is significant in the control loop of the HMC. Thus, according to Requirement 4 it is enough to consider a single pole. The resuling formula is:

$$Q(i,s) = \frac{\sqrt{\Delta P} * K_1 * i}{\frac{s}{\omega} + 1} \tag{6.5}$$

where

Q =	Output oil flow [L/s]
ΔP =	System pressure [bar]
K_1 =	Servo valve gain factor [1]
ω =	Natural frequency [s^{-1}]
i =	Input current [A]

Implementation

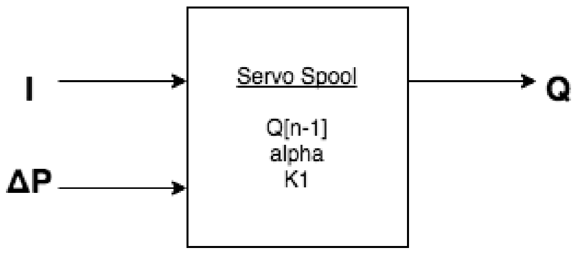


Figure 6.10: Block diagram of servo simulation object

Since the HMC05 outputs the servo current, this can be considered an input for the simulation. The simulation must calculate the resulting oil flow, since this is dependant on the system pressure, this is also considered an input of the servo simulation block.
The single pole model has been linearized with the exponential smoothing factor. This decreases computation time, since the step size is very small, this does not introduce errors. The in-output relationship of the servo is:

$$Q_{n+1}(I, \Delta P) = \alpha * (\sqrt{\Delta p} * I * K) + (1 - \alpha) * Q_n \tag{6.6}$$

where

$\alpha =$	$0 < \alpha < 1$ and $\alpha = 1 - e^{\frac{\tau}{\Delta t}}$
Q =	Outgoing flow (going to the positioning cylinder) [L/min]
K =	Servo gain factor

6.2.3. Positioning cylinder

This cylinder directly controls the swashplate angle (and consequently the output torque). This angle can be adjusted by moving the cylinder. The cylinder has oil on both sides and can be moved by displacing oil from both sides. The positioning cylinder is typically connected with a 4-way electro-hydraulic servo that has two outputs ($A = -B$).

Model

$$\dot{x} = Q * A_{cyl} \tag{6.7}$$

where

x =	position of cylinder [m]
Q =	Oil flow that displaces cylinder [$\frac{L}{s}$]
$A_{cyl} =$	cylinder cross area [m^2]

Essentially this component integrates the oil flow, that controls the swashplate angle.

6.2.4. Implementation

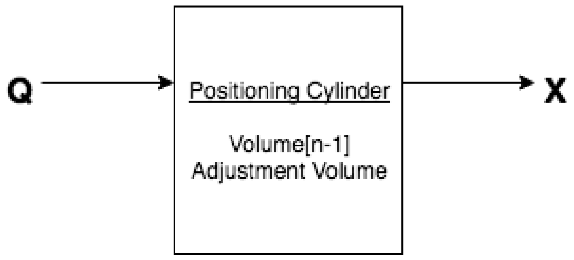


Figure 6.11: Block diagram of positioning cylinder simulation object

Since the oil flow that displaces the cylinder is given by the servo, this is the input of the simulation block. The simulation block must compute the resulting cylinder position (based on the old cylinder position) by integrating the incoming oil flow. However, saturation must be implemented as well in the model.

$$x_{n+1}(Q) = \begin{cases} -1 & \text{for } x_n \leq -1 \\ x_n + Q * \Delta T/(2 * V_{tot}) & \text{for } -1 < x_n < 1 \\ 1 & \text{for } x_n \geq 1 \end{cases} \tag{6.8}$$

6.2.5. Logic Valve

The logic valve is an electro-hydraulic solenoid valve. This valve has been added as a safety feature. In standby the hydraulic plant should be stationary and secure. For extra safety, the system pressure ΔP is not directly connected to the hydraulic motor, but rather connected through a Logic valve. This binary valve is *normally closed.* If the HMC provides a current, the logic valve will close and the motor will be subjected to system pressure. A spring is installed in the logic valve to assure the valve is opened when no current is supplied.

Model

The opening and closing of the valve has a significant delay.

The time delay depends on the pressure difference and the spring constant inside the valve. Apart from the latency, the actual opening and closing times can be considered instant (hysteresis).

implementation

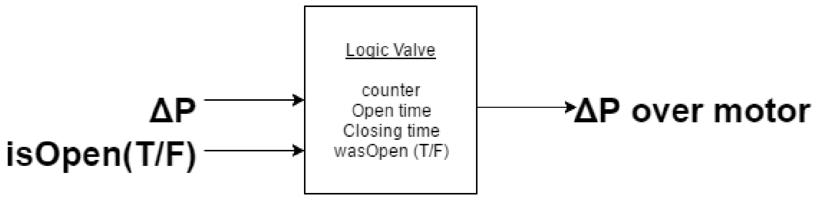


Figure 6.12: Block diagram of logic valve simulation object

Since commissioning engineers measure and observe the opening and closing time (typically around 20ms and 200 ms respectively), the model will have these parameters as configuration.
This simulation triggers a counter if the old valve position differs from the input valve position. When the counter reaches the opening or closing time, it toggles the output between 0 bar and the input pressure.

6.2.6. Backup Servo valve

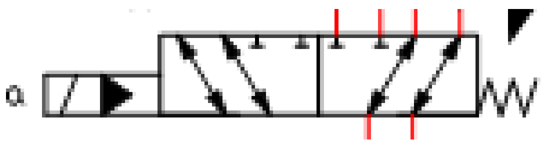


Figure 6.13: Hydraulic solenoid valve

For safety reasons one backup servo is present. The backup servo can bypass the regular servo and apply an oil flow to the positioning cylinder and can be used in an emergency situation. The backup servo and the regular servo are both connected to a switch/valve. This switch *normally* connects the backup servo to the positioning cylinder. By applying current to the input of the switch, the regular servo is attached to the positioning cylinder.

Model

Similarly to the binary valve, the backup servo switch also experiences latency due to hysteresis. This is modelled with a time delay.

Implementation

Since the switch is connected to a servo (a flow source), both inputs are an oil flow. The output is either equal to Q_{backup} or $Q_{regular}$.

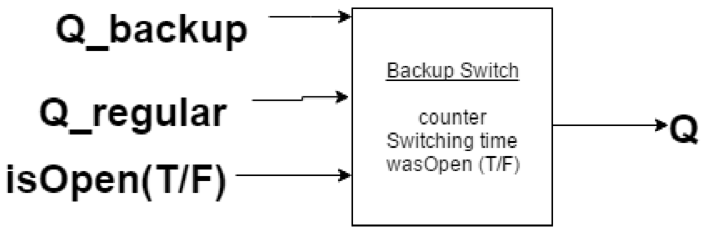


Figure 6.14: Block diagram of backup servo switch

6.2.7. Hydraulic-mechanical brake

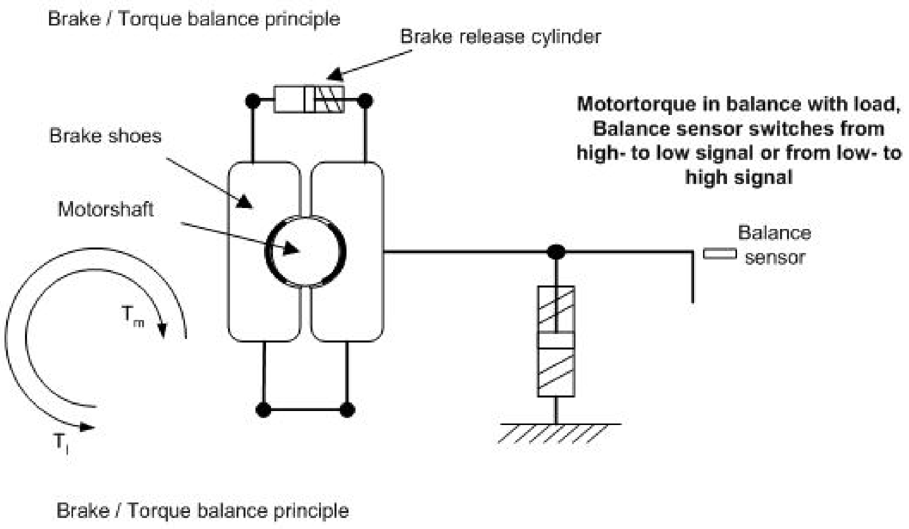

Brake / Torque balance principle
Brake release cylinder
Brake shoes
Motortorque in balance with load, Balance sensor switches from high- to low signal or from low- to high signal
Motorshaft
Balance sensor
T_m
T_l
Brake / Torque balance principle

Figure 6.15: Cross section of motorshaft illustrating the brake shoe, the release cylinder and the torque balancing sensor

This component is the mechanical brake. When the HMC05 goes idle, this brake shoe is dropped on winch shaft. The braking system is essentially a cylinder with a spring system. If enough oil is displaced and enough pressure is built up in the cylinder, it displaces and lifts the brake. The braking system is therefore *normally* braking. Inside the braking cylinder there is usually a pressure sensor.
Note: The braking system is not directly connected to the system pressure, but rather to a lower pressure (typically around 70 bars).

Model

The brake can be modelled as a binary hydraulic valve followed by a cylinder (integrator with saturation). Figure 6.16 shows a typical time versus pressure graph. The opening and closing time are modelled as a delay, the increasing and decreasing pressure is linearized.

Implementation

The simulation block of the braking system receives the brake valve state from the HMC05. Since the HMC05 expects to measure the pressure inside the braking cylinder, this must be an output of the simulation block. The time necessary to fill the braking cylinder (and lift the brake) as well as the pressure it is connected to, is the output.

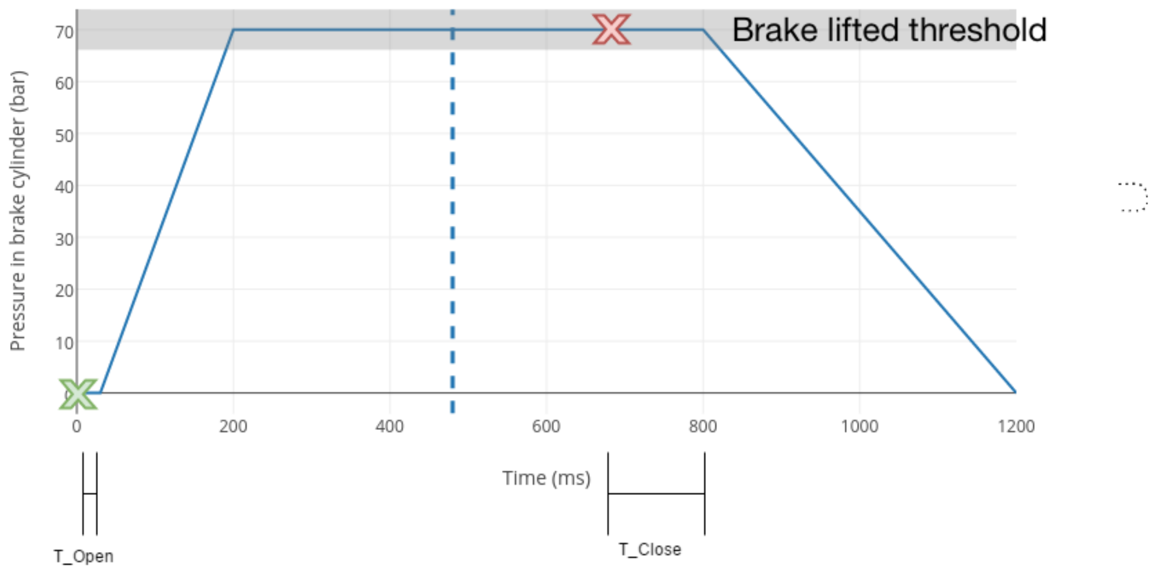


Figure 6.16: Time vs brake pressure graph. Valve opening and closing times visualized. Linear increasing and decreasing pressure up to max pressure in breaks (70 bar)

6.2.8. Torque Balancer

As mentioned before, the mechanical brake is dropped on the winch shaft as soon as the HMC05 goes idle. This is typically a few seconds after no new set-point is given.
When the brake is lifted, the HMC05 may experience control trouble if it cannot observe the load torque. Therefore **optionally** a torque balancer can be installed. This is a sensor that can observe if the torque supplied by the motor is matched with the load whilst the brake is still applied.
The HMC05 observes the torque balancer as a binary sensor (torque balanced: true/false).

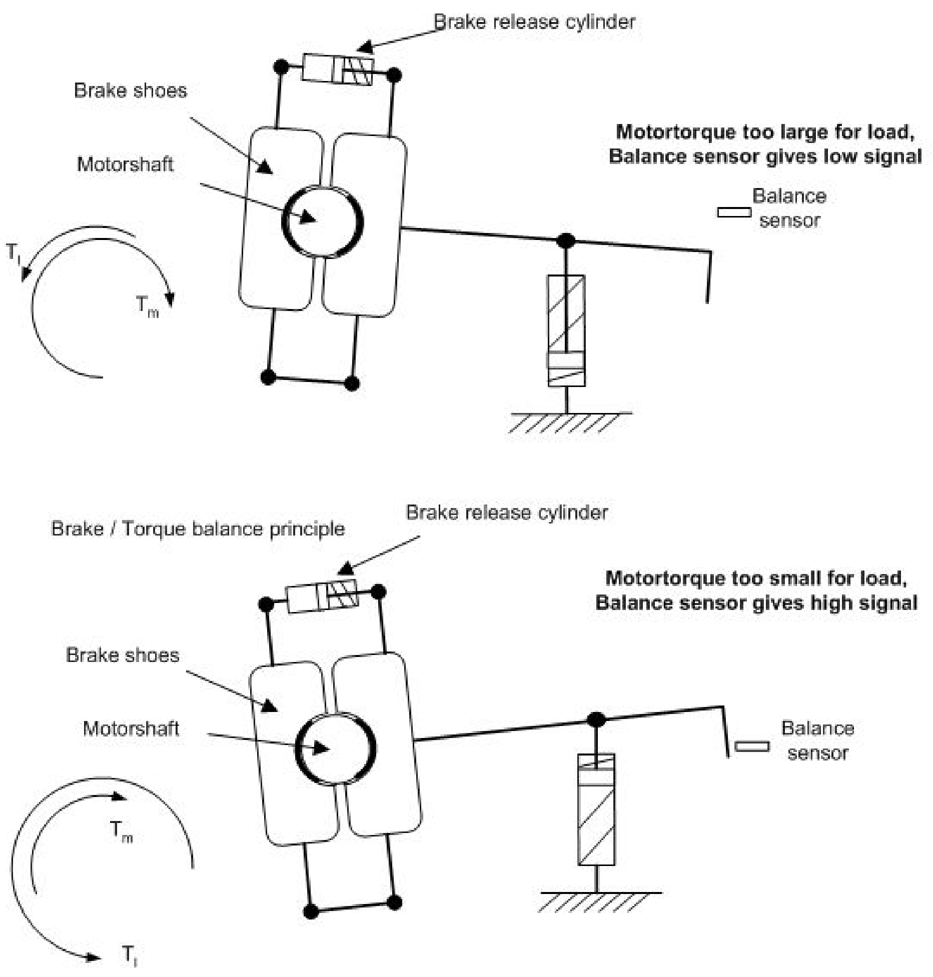

Brake release cylinder
Brake shoes
Motorshaft
Motortorque too large for load,
Balance sensor gives low signal
Balance sensor
T_l
T_m
Brake / Torque balance principle
Brake release cylinder
Motortorque too small for load,
Balance sensor gives high signal
Brake shoes
Motorshaft
Balance sensor
T_m
T_l

Figure 6.17: Torque balancer principle

Model

$$\text{isBalanced} = \begin{cases} True & \text{for } \mid T_{load} - T_{motor} \mid < T_{th} \\ False & \text{for } \mid T_{load} - T_{motor} \mid \geq T_{th} \end{cases} \tag{6.9}$$

where

T_{th} = Threshold torque [N*m]

6.2.9. Torsion spring

A torsion spring is the rotary equivalent of a linear spring. It can store potential energy and convert it back to (rotary) kinetic energy.

Model

$$\tau = -\kappa * \theta \tag{6.10}$$

where

τ = Torque exerted by spring [N*m]

κ = Torsion spring constant

θ = Angle of twist with respect to equilibrium [rad]

Implementation

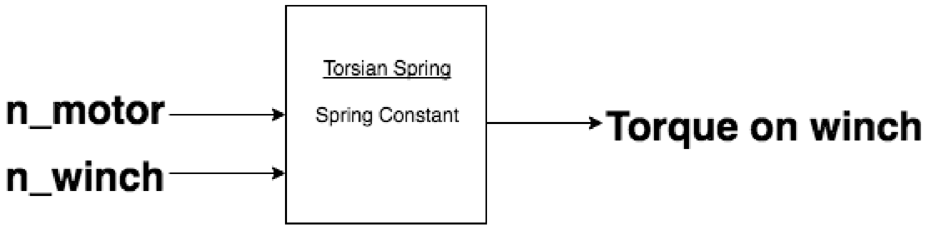


Figure 6.18: Block diagram of torsian spring simulation object

This component must be implemented with integrating causality. The simulation block should therefore output a torque based on the difference in angular speed between the two connected shafts. The component connects the motor shaft with the gearbox or the winch shaft.

6.2.10. Winch

A winch is a mechanical cylindrical shaped component that stores unused length of cable. It has a large moment of inertia due to its high mass and size. Additionally, the cable mass that is stored in the winch can contribute a significant amount to the moment of inertia. Therefor the moment of inertia does depend on cable position.

Model

The priority to implement a variable moment of inertia is considered low (from HedoN). The current model considers a constant moment of inertia (I). The structure provided for the simulation does however allow for this feature to be added in the future.

$$\tau = I * \ddot{\theta} \tag{6.11}$$

where

$\tau =$ Applied torque $[N * m]$

I = Moment of inertia $[kg * m^2]$

$\theta =$ Angular position [rad]

$\ddot{\Theta} =$ Angular acceleration $[\frac{rad}{s^2}]$

Implementation

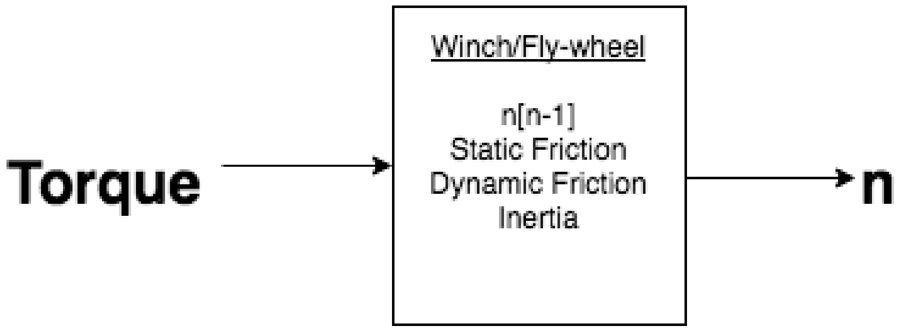


Figure 6.19: Block diagram of winch simulation object

To maintain integrating causality, the input of the winch simulation block is the torque and the output is the rotary velocity

6.2.11. Hydraulic Pump

In secondary hydraulics the system pressure must be constant. The hydraulic pump is an oil pump that is pressure controlled. There is no fundamental difference between a hydraulic motor or pump, the difference being that an hydraulic pump receives rotational energy from electric motors.

Model

The hydraulic pump is parametrized by a commissioning engineer in such a way that within a range of flow system demand the pressure remains constant (as the feedback loop meets its boundary conditions). If the system load is higher than the maximum oil flow, the pressure will drop. Alternatively, if hydraulic motors are recuperating energy (eg. potential energy from heavy mass), the pressure may build up.

$$P(Q_{req}) = \begin{cases} > \Delta P & \text{for } Q_{req} < Q_{min} \\ \Delta P & \text{for } Q_{min} < Q_{req} < Q_{max} \\ < \Delta P & \text{for } Q_{req} > Q_{max} \end{cases} \tag{6.12}$$

where

Δ P = System pressure [bar] = $P_{high} - P_{boost}$

Q_{req} = Oil flow demand from hydraulic system [L/s]

Q_{min} = Minimum oil flow for controlled pressure. $Q_{min} \leq 0$ [L/s]

Q_{max} = Maximum oil flow for controlled pressure [L/s]

Implementation

The hydraulic pump has been combined with the accumulator. The implementation of the hydraulic pump and accumulator is discussed in the next subsection.

6.2.12. Accumulators

The accumulator is an **optional** component that can store energy. An accumulator is typically used to store energy for sea wave compensation (active heave compensation, constant height with respect to seabed), since the period of such waves is reasonably small. The use of an accumulator can reduce the need of more hydraulic pumps. The energy is stored in pressurizing nitrogen in big tanks. The energy stored in an accumulator can be utilized rapidly.

Model

Previously it was shown that outside a boundary condition, the hydraulic pumps cannot maintain constant pressure. In such case the system pressure is dictated by the stiffness of the system (eg. expansion of oil tubes). If an accumulator is present, this component will dominate and determine the system pressure. The accumulator therefor supplements the above formula of the pumps.

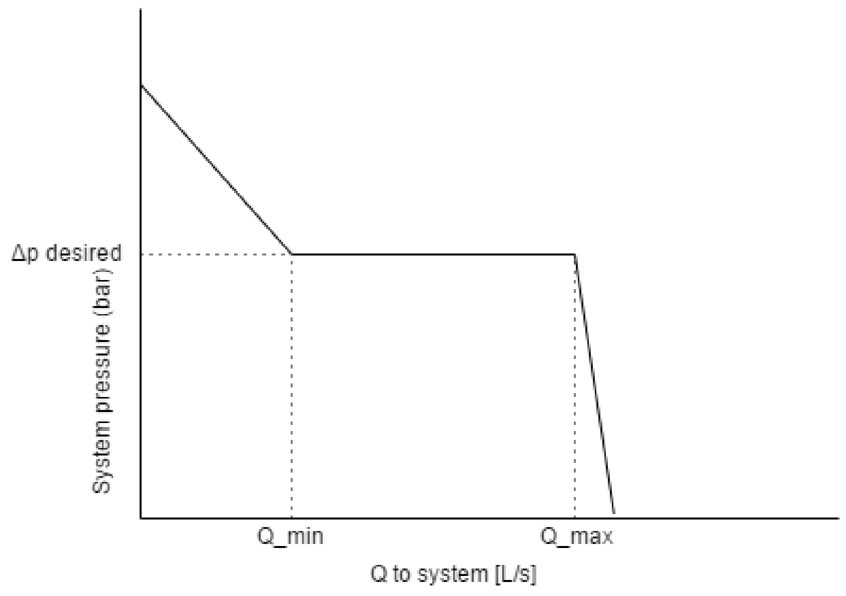


Figure 6.20: System pressure vs oil consumed graph

$$P(Q_{req}) = \begin{cases} \Delta P + (Q_{min} - Q_{req}) * K_1 & \text{for } Q_{req} > Q_{max} \\ \Delta P & \text{for } Q_{min} < Q_{req} < Q_{max} \\ \Delta P + (Q_{max} - Q_{req}) * K_2 & \text{for } Q_{req} < Q_{min} \end{cases} \tag{6.13}$$

where

K_1 = Linear pressure decreasing constant

K_2 = Linear pressure increasing constant

K_1 and K_2 can be determined based on the system pressure, nitrogen pressure, nitrogen volume and temperature.

Implementation

The accumulator and pump are implemented together in a single simulation block. The in and outputs of this block, as well as the configuring parameters can be seen in Figure 6.20.

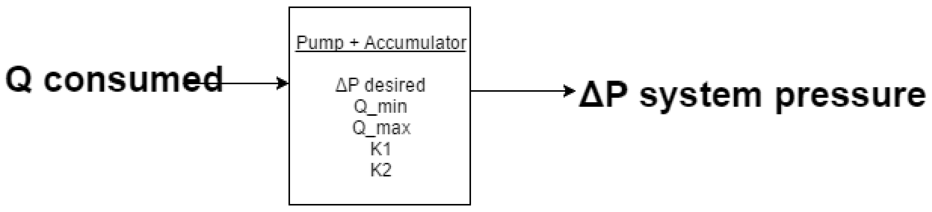


Figure 6.21: Block diagram of the pump and accumulator

This simulation block can calculate the system pressure P_{n+1}, by summing all the $Q_{consumed} = Q_{motor} + Q_{servo} + Q_{brake}$.

6.3. Multiple motor setups

Thus far, a single typical plant has been discussed. All parameters, such as the motor capacity (typically expressed in maximum oil displacement[cm^3], winch inertia and load can easily be configured. However, the HMC can support a number of different setups, namely multiple motor setups. This chapter will explain the different variations possible and their implementation in the existing simulation framework.
To achieve more kinetic power, multiple axial piston motors can be configured to work in parallel. A single

HMC can support up to four hydraulic motors (and their corresponding valves, servos, encoders, etc.). Four HMC's can be combined to form a single controller that can control up to 16 motors. Generally speaking, a multiple motor setup is constructed by copying a number of typical single motor setups. However, a couple of distinctions must be made:

1. All motors have their own logic valve and bypass servo valve.
2. A **single** backup servo is available for all motors. All bypass valves are connected to the same backup servo.
3. A single mechanical braking system is present.
4. Mechanically the motors must all be connected to the same winch, this can either be done with a single gearbox or each motor can have its own gearbox.
5. Not all motors must have an encoder. The more encoders are present, the better the HMC can control the system.
6. Multiple motors greatly decrease the stiffness of the system. The torsion springs in the system increase. This can lead to resonances, only measurable if each motor has an encoder.

All different setups can be simulated with the existing blocks. The simulation file can be configured by a simple text file. This text file must describe which motor has an encoder and how many gearboxes are present.

6.4. Full simulation

In Figure 6.22 all simulation blocks have been interconnected to form a complete simulation of a winch based hydraulic plant. The HMC provides the servo current and the current to the valves (blue), the HMC expects to receive back the brake pressure, system pressure and rotary speed (encoder. This is shown in red.

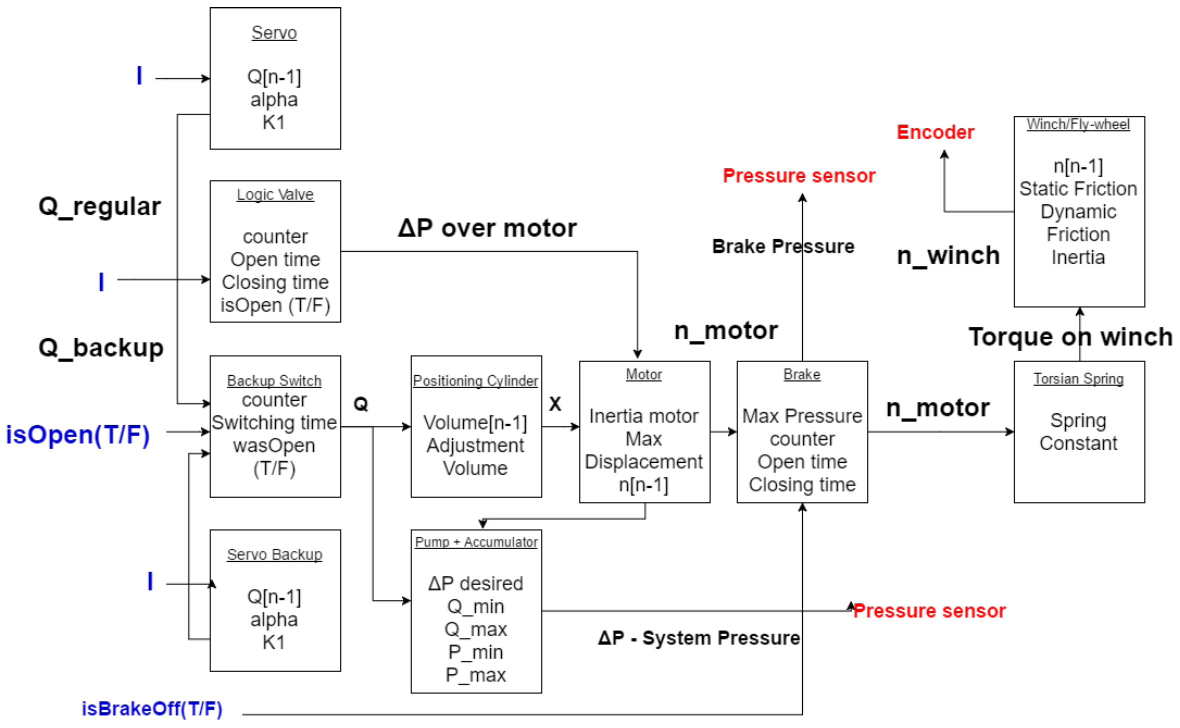


Figure 6.22: Full simulation overview. Blue are simulation input. Red are simulation output.

7 Monitor System

7.1. Role

The monitoring system consist of an external ethernet connected monitoring computer which runs a database for storing data and contains a web interface for visualization and control. This fulfills Requirement 5. Requirement 6 states that the simulation system also has to work without a monitoring system. This requirements is fulfilled by using an external computer which is on different ethernet network then the hardware IO units.

In theory it would be possible to run the visualization and database on the SAMA5D3 microprocessor. Visualization will produce IO instructions which tend to be slow. Similarly database managers have to write to high-level memory which leads long processor stalls. As those processes run on different threads on the same processor and a single task (update and a webpage and the database, writing data to non-volatile memory) typically takes more then 1ms this would endanger strict 1ms tick of the system.

7.2. Configuration

The same configuration file that is used on the HSU itself to generate a simulation. With this file the monitoring system is able to create a database with context and store understand content of the data send by the HSU. No separate initialization between the HSU and the monitoring system is needed because the monitoring system works independently.

7.3. Connection to the monitoring system

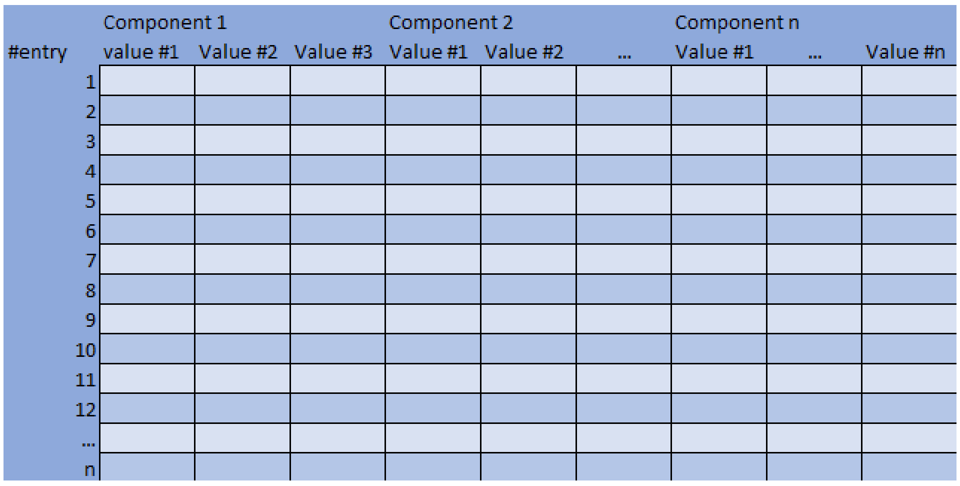
The connection from the monitoring computer to the HSU will be an ethernet level 2 connection like the one used to communicate with the hardware IO units. Timing is a less relevant factor in this case, hardware IO units have to react during the same millisecond, no feedback is received when data is send to the monitoring computer. However it is relevant to minimize the amount of time it takes prepare and send a packet to the monitoring computer because this happens in the 1ms loop of the HSU. A level 2 connection has less overhead then a higher level protocol as explained in Chapter 8.1 and is thus faster prepared. The management of data which is send data which will be send to similar the one used in sending data to the hardware IO units.

7.4. System environment

The monitoring system will be running on 'full OS' on a normal computer. No real-time has no real-time criteria and must run an Database Management System (DBMS) and webserver software, this is easiest on a full OS. The monitoring computer will run software which receives and unpacks the ethernet packets and writes them to the database.

7.5. Data storage

A very basic database structure will run on the monitoring system. This database extremely simple and will be build using the configuration file as this defines all values which are send by the HSU to the monitoring computer and the place in the packet. The database will get 1 packets for each millisecond. The table will



	Component 1			Component 2			Component n		
#entry	value #1	Value #2	Value #3	Value #1	Value #2	...	Value #1	...	Value #n
1									
2									
3									
4									
5									
6									
7									
8									
9									
10									
11									
12									
...									
n									

Figure 7.1: Structure of the database on the monitoring computer

look like Figure 7.1. Every entree represents a milliseconds. Values of the hardware IO units are displayed after this.

7.6. Visualization

Visualization in on the monitoring means showing graphs. The data pushed to the database can be plotted against the time. This shows how to hydraulic system reacts. The contect of the data the monitoring system gets from the configuration file (Chapter 7.2).

8
Integration

All parts of the system have been described separately. This chapter will integrate all this components to one the final system and explain typical use of this system.

The modules of the system are:

- Ethernet communication
- IO interpreter and data structure
- Simulation
- The monitoring system

8.1. Ethernet communication

All elements in the simulation system are connected using ethernet connection. The done on a low layer to keep the overhead of the system low. This results in less overhead in and faster communication, this is can be done because the simulation network is relatively small and is not connected to the global network. The HSU is master in the communication with hardware IO modules. It communicates with the hardware IO units using broadcast signals, the hardware IO units react with packet send to the HSU. Before the simulation starts the HSU will first initialize all hardware IO units in order to check if the system is connected properly.

8.2. IO interpreter and data structure

In order to communicate efficiently between hardware IO units via the communication and the simulation a software layer which acts as middle layer is created. This middle layer handles in- and outgoing data and efficiently writes back calculated values from the simulation in order to minimize the time it takes to pack and unpack packets.

In the initialization phase of the simulation system the IO interpreter reserves blocks of memory for the hardware IO modules. Every IO module got it own reserved space in the memory. This reserved places are placed after each other such that the send system only has to know the first byte and the length of the message in order to broadcast the message to the IO modules. The IO modules know from the initialization phase were to start reading in the packets and how many bytes are reserved for them.
The structure for sending data back is known by both the IO modules and the HSU. When the HSU receives data back from the IO module, the HSU can calculate the place were to write the received data on a similair way as the send block is structered.

8.3. Simulation engine

The simulation engine simulates and hydraulic plant which need to be controlled by an HMC05. The simulation consist of blocks which contains models of physical parts in an hydraulic system such as valves, encoders, motors, springs and loads. There exist three types of simulation blocks:

figures/integration/config_file.png

Figure 8.1: The configuration file of the simulation system

Inputs : Need values from the IO modules, ask the manager were values are stored in the memory, calculates a value gives this value to the simulator

Outputs : Get values from other blocks, calculate a value and gives this value to the manager which will send it to the IO hardware module.

Intermediate : Get values for the simulator, calculas

All calculates done represent a simulated object. The simulation engine is based on the same hydraulic models as the HMC05 itself.

8.4. The monitoring system

This is an optional component in the HMC05 testing system. When it is connected it is used to show graphs to the user about the system performance. Every itteration the HSU sends data to the monitoring system which the monitoring system stores in a database. When HSU is running its is possible to simulate a broken cable by sending an instruction to the HSU.

8.4.1. Configuration file

The monitoring system can also be used to specify different hydraulic systems. As discussed in the example of the data structurer and IO interpreter (specifically Table 5.1) as well as in Section 6.3 about multiple motor hydraulic plants, a number of things must be specified and configured for the simulation system.
The configuration must contain information about:

- Specify the number of axial piston motors and all the parameters of each sub module. (ie. motor cc, positioning cylinder, LVDT type etc).
- Specify the observation points: Where are all the pressure sensors. How many encoders are present.
- Specify for each mac address the purpose of that I/O hardware module (table 5.1). For example: mac address 20 is the servo simulator hardware module of the third motor.

The configuration file completely defines all loose variables of the complete simulation system. Both the HSU and the monitoring computer must have the same configuration file. The monitoring system can create a database structure based on the configuration file (Section 7.2), while the HSU can feed all parameters to the simulation module, as well as create all the IO interpreters and attach them to the data structurer. An example of how such an datafile would look like can be found in figure 8.1. In this case it is an CSV based file with consist out of an setup part which describes the motor setup. The motor setup is very similar among different configurations and the physical addresses to which they are connected in the network. This is done to place dip switches on the hardware IO units.

8.5. System routine

When the system is started it starts in the initializing phase before it goes to the main simulation loop. The is visualized in Figure 8.2. After the initialization phase the main loops starts which is forced on 1ms.

8.5.1. Initialization phase

The initialization phase is in pseudo Code 8.5.1. The loop is visualized in figure 8.3.

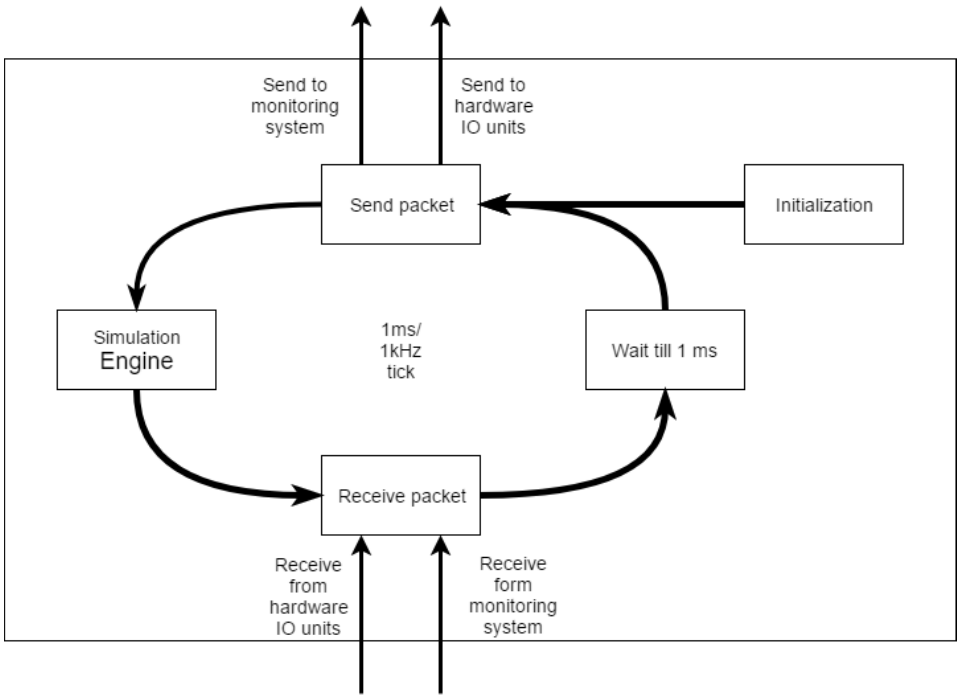

Send to monitoring system
Send to hardware IO units
Send packet
Initialization
Simulation Engine
1ms/ 1kHz tick
Wait till 1 ms
Receive packet
Receive from hardware IO units
Receive form monitoring system

Figure 8.2: The routine of the HSU

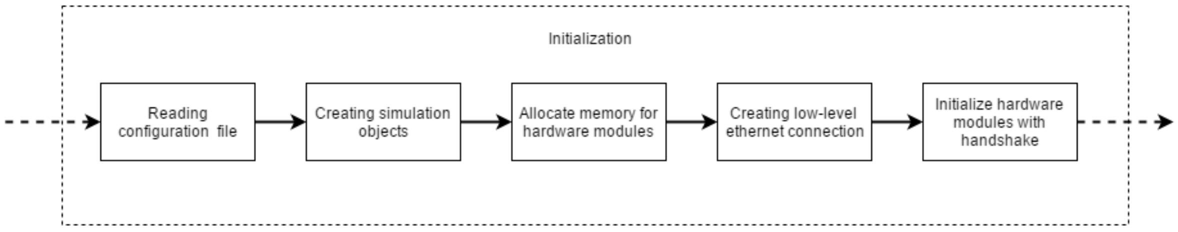

Initialization
Reading configuration file
Creating simulation objects
Allocate memory for hardware modules
Creating low-level ethernet connection
Initialize hardware modules with handshake

Figure 8.3: Initialization phase

```
begin
    Read configuration file();
    Create Simulation Object();
    Reserve Memory for IO Hardware Components();
    Prepare Ethernet Sockets();
    Initialize Hardware Units using Handshake();
    Start Central Clock();
end
```

Algorithm 1: Initialization phase of simulation system

The initialization got the following functions which function can be found in thesis:

- **Read configuration file**: Configures the system before use, discussed in chapter 8.4.1
- **Create simulation objects**: Creation of simulated objects, discussed in chapter 6.1.2
- **Reserve Memory for IO Hardware components**: reserveres memory for IO of the HSU. Discussed in Chapter 8.4.1
- **Prepare Ethernet Sockets** Prepare ethernet for fast and effficient communication. Discussed in Chapter 8.1
- **Initialize hardware Units using Handshake**: discussed in Chapter 5.4.2

The role of the central clock will be discussed in Chapter 8.5.4.

8.5.2. Iteration loop

The iteration loop is in pseudo Code 8.5.2

```
begin
    Initialization();
    while Not end simulation do
        Send configuration to IO hardware modules();
        Send simulation data to monitoring system();
        Simulation();
        Check Instructions Received from monitoring system();
        while Not 1 ms do
            Check and handle received packets from IO hardware modules();
        end
    end
end
```

Algorithm 2: HSU main iteration loop

After the initialization function the system comes in the main loop. At the begin the configurations are send to the monitoring system and the IO harware modules. After this the hydraulic plants is simulated using values it received in the previous iteration. After that it wait till the 1 millisecond before it the loop starts again.

8.5.3. Main loop

The simulation exists of a big amount of simulation objects. These objects are created in initialization phase and all have an input and an output. As the system is build in causal fashion the output values can be calculated sequential. The motor has inputs and calculates step for steps the output values. The main loops calls all function which together simulates the complete motor. This simulation is in psuedo code descibed by 'simulation()'. The objects which are simulated in the current setup can be found in pseudo code 8.5.3.

```
Q = spool.computeFlow(Qnorm);
x = cylinder.computePosition(Q);
brakepressure = brake.computePressure(breakCurrent);
if $b$ then
    r
end
akepressure > 0.8 toerental = motor.computeSpeed(x, load.computeTorque());
else
    t
end
oerental = 0;
T = spring.computeTorque(toerental, n);
n = winch.computeAngularVelocity(T);
oilUsed = Q + motor.Q;
pressure = pumpAccu.computePressure(oilUsed);
```

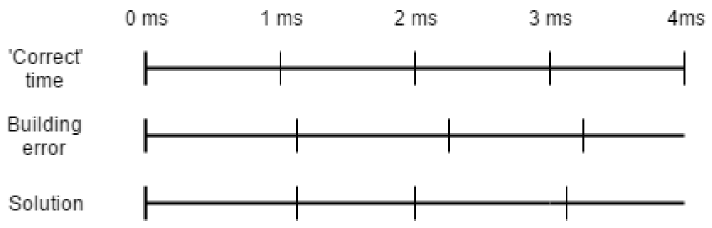


Figure 8.4: Timing problems in the system

8.5.4. Strict timing

The system has an strict 1ms tick. As is already argued in Chapter 3.2 processors are deterministic in nature. If each simulation loop would keep its own time this would lead to an ever increasing error because the time a loop takes is always $\geq 1ms$, as a check which is performed is only true when the time difference is bigger then 1ms. This is visualized in Figure 8.4 situation 2. The average time would be a little bit more then 1ms. This problem is solved by starting a timer at the beginning of the simulation loop and compare each loop with this starting time. The will make the average tick 1 millisecond. However the actual time each loops endures will vary around 1 ms.

9 Testing and results

9.1. Introduction

In this chapter the tests and the results of each module will be discussed. The Ethernet module, the data structurer & IO interpreter and the simulation have been tested. The components have also been combined and tested in an integrated system test. Due to the lack of IO hardware modules, the simulation system has never been tested in combination with the HMC05.

9.2. Ethernet Module

The Ethernet module must be able to send unicast and broadcast packages and must forward incoming packages from the receive buffer to the data structurer. To test if the Ethernet module creates the correct headers for outgoing packets, Wireshark has been used. Wireshark is a software program that can intercept packages and analyze the network protocols and payload. The test results show that the Ethernet module can receive Ethernet packages and forward the payload to the data structurer. The test also shows that the Ethernet module can send broadcasts and unicast packages with any payload originating from the data structurer. The thesis of Sam & Jorden [4] contains additional (timing) test results about communication between the hardware IO modules microprocessors and the HSU.

9.3. Data structurer and IO interpreter

To test the data structurer module a large number of hardware components on a switch are desired. Since the data structurer identifies each hardware component by their unique mac address, one would need to connect a number of different devices to a switch. Since none of the IO hardware modules were available and only two IO hardware microprocessor developer boards were provided by HedoN, a new setup was created.

9.3.1. Hardware IO modules spoofing

This setup is centered around the fact that in Python when creating a raw Ethernet socket on Linux, the OSI level 2 header is not created by the library itself. This is not the case for the Linux C++ library (see the limitation Section 4.6 of the Ethernet chapter). Since the header includes the source mac address, this implies that in Python one can spoof Ethernet packages and make them seemingly originate from non existing additional hardware. This feature is used to simulate many components for the HSU, using a single computer.

9.3.2. Test setup

The goal of this test setup is to determine if the data structurer is correctly implemented. This is done by configuring a large simulation system (with as many as 150 IO interpreters and hardware modules, each with a unique mac address) and by connecting the SAMA5 chip to a Linux computer running a Python script. The Python script is programmed to echo back an ACK by copying the arriving destination mac address and sending back an Ethernet packet with that address as the source, essentially impersonating a hardware component. The Python script is programmed to keep a list of all the mac addresses it has spoofed and which offset is associated to that address. This list is compared with the *original* list of the HSU data structurer component.

9.3.3. Results and interpretation

The test results show that the data structurer can easily scale to 150 components. Each component on the network receives the correct offset imposed by the HSU.

The limiting factor found in the data structurer is the broadcast package size. With an increasing amount of components, the broadcast size can reach the maximum Ethernet package size of 1500 bytes [12]. However according to Requirements 1 and 2 the simulation system IO are limited to that of the HMC, which is maximum of about 70 components.

9.4. Hydraulic simulation

Even though the hydraulic plant simulation cannot be connected to the HMC, the HMC behaviour can be simplistically simulated and the hydraulic plant can be evaluated.

9.4.1. Replacing the HMC

The most relevant HMC output is the servo current. Recall that this output influences the winch speed (with a double integral relationship) and can rapidly destabilize the winch. Based on the model one can make predictions of the behaviour of the hydraulic plant under certain servo current. A number of different tests have been run.

1. Short servo current pulses.
2. Linear increasing servo currents.
3. Linear increasing servo currents with brake applied.

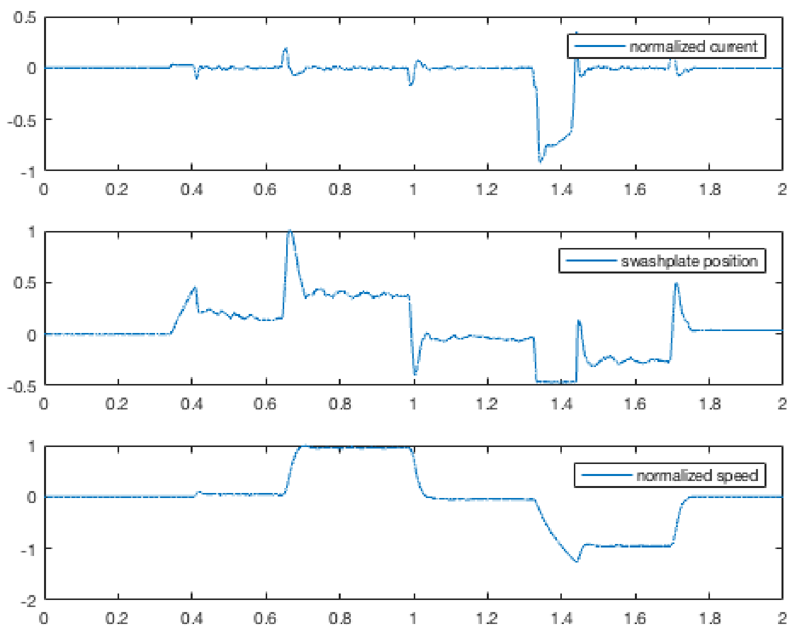


Figure 9.1: Real world HMC measurements. A very small increase in current causes a change is swash plate position and increases the winch speed. All measurements have been normalized.

Figure 9.1 shows measurements obtained from the HMC of a real world hydraulic plant. The figure shows that a very small current can cause a big swash plate position swing which can accelerate the winch.

9.4.2. results

In the first simulation the logic valve, brake and bypass servo were all switched on (ie. *normal* operation). Figure 9.2 shows the double integral relationship between the input current and the output winch speed. The HMC servo current

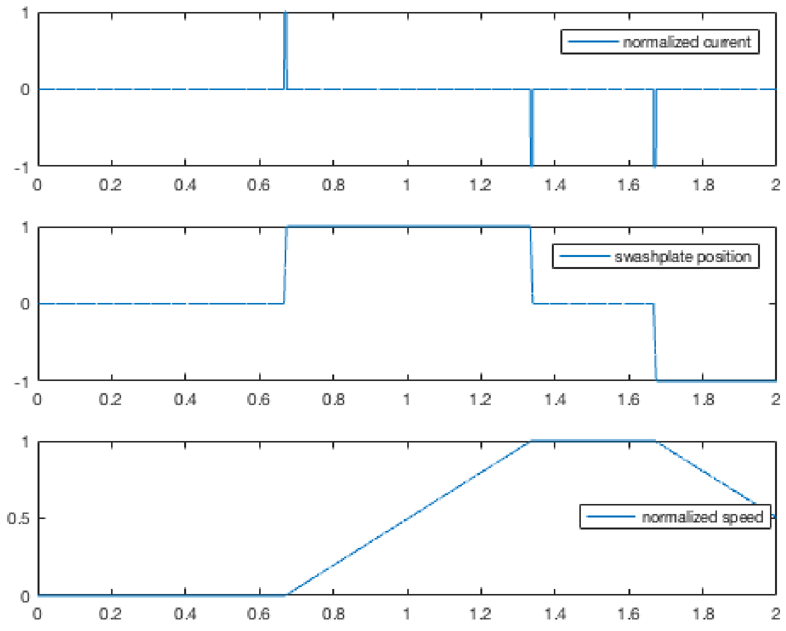


Figure 9.2: Hydraulic simulation results with pulse servo currents as input.

In the second and third simulation the servo current had a triangle shape. In the second simulation the brake is lifted, whereas in the third simulation the brake was applied at t=0. At t=1s the fictional HMC lifts the break shaft and the winch starts to move.

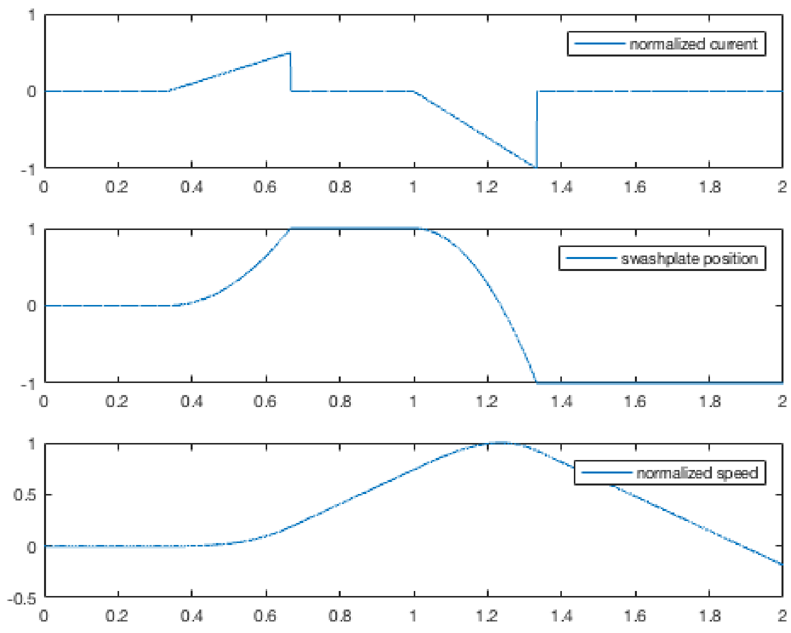


Figure 9.3: Hydraulic simulation results with triangle shaped servo input current. No brake applied.

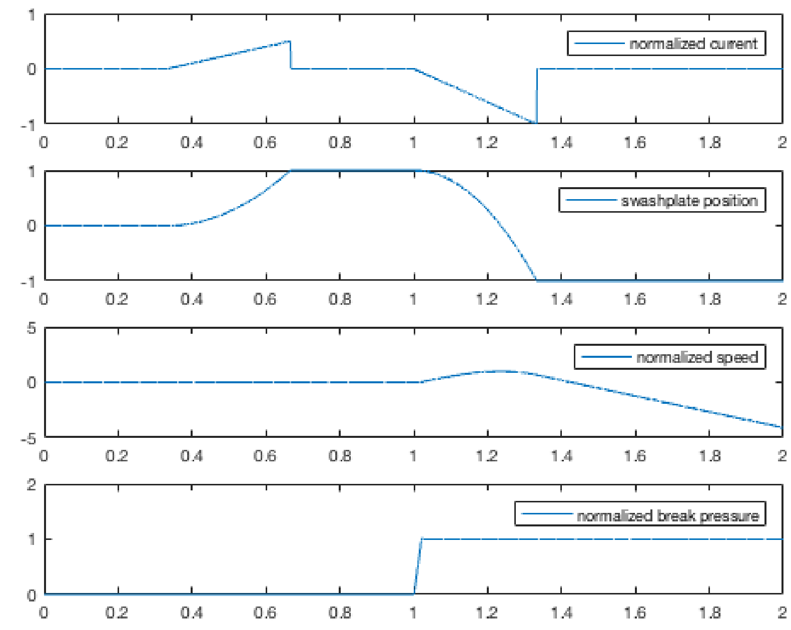


Figure 9.4: Hydraulic simulation reults with triangle shaped servo input current. Between t=[0,1] the brake is applied.

9.4.3. Interpretation

Since there is no feedback between the servo current and the cylinder position or winch speed, the results coming from the simulation are hard to compare with the real world measurements with a HMC. The simulation results do show however the double integral relationship between servo current and winch speed. This is in line with the model and validates the implementation of the simulation in causal software objects.

9.5. Hydraulic Simulation 2

A second test is done to test the performance of the hydraulic simulation. Each test endured 10000 iterations. The total simulation was compared against a simulation which kept getting more difficult by adding more motors.

9.5.1. Testing conditions

The runned on the SAMA5D3 while the send and receive functions were disabled. This meant that only the raw simulation time was measured.

9.5.2. Limitations

The amount of calculations per motor is relatively small. As the simulator gets extended to a more precise models the calculation time will rise.

9.5.3. Results

Figure 9.5 shows the speed as more motors are added. The blue line shows the average of 5 independent measurements that were done.

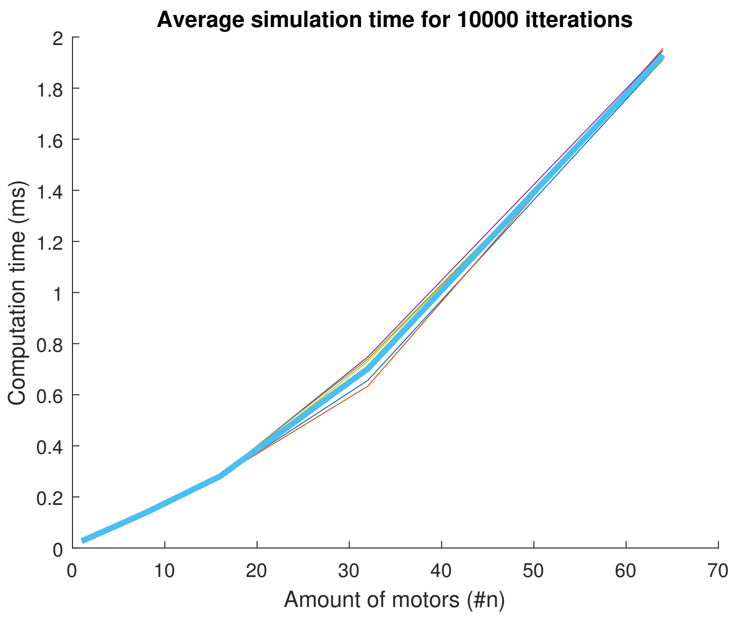


Figure 9.5: Simulation speed compared to difficulty

9.5.4. Interpretation

- The simulation doesn't scale completely linear when more motors are added
- Around the 40 motors the simulation alone takes more then 1 ms.
- As more motors are added the spread in computation time rises

9.6. Integrated system

Even though no hardware IO module are present, a developer board of the microprocessor embedded in the pindriver (the E70) was available. A potentialmeter was attached to the ADC of the E70. In this test the data flow between the different components of the HSU is tested and validated. It is also tested if the system loop is constant on 1ms.

9.6.1. Testing conditions

Three devices are connected to an Ethernet switch: The HSU, a simplified hardware input module (the E70 development board with a potentiometer) and a Linux computer with Wireshark running.
The E70 development board is impersonating a pindriver. The potentiometer resistance is sent to the HSU as a 'current measurement' of the pindriver. The HSU is configured such that this is interpreted as a servo current originating from the HMC05. All other inputs of the HSU were fixed as constants (the brake was lifted, the motor logic valve was closed, bypass valve active).

This test can illustrate if the datapath between an analog measurement (the resistance of a potentiometer) and the simulation engine is correct. It can also show if the resulting $t = t_0 + \Delta t$ values from the simulation engine are pushed to the hardware modules, by analyzing the broadcast package with Wireshark.

9.6.2. Test limitations

This test can determine if the datapath of the simulation system is correct. However since only a single I/O hardware component is present, the timing results may not be representative for a simulation system configuration with more I/O hardware components, because the time to unwrap Ethernet packets and to interpret the measurements is dependant on the amount if I/O hardware components present in the simulation system.

9.6.3. Results

The following results are obtained by running the system loop 10.000 times. The configuration of the hydraulic simulation system is adjusted to simulate either 1, 4, 6 or 16 motors (16 motors is the maximum amount of motors any HMC05 setup can control). It is noted that the fictional servo current from the potentiometer is the input for all the motors.
The Wireshark analysis in combination with console outputs of the HSU shows that the datapath within the simulation system is correct.
In Table 9.6.3 the system loop timing results are shown.

#n motors	1	4	8	16
Longer then 1,05ms	1.15%	1,10%	1,13%	1,16%
Shorter then 0,05ms	2,55%	2,29%	2.46%	2,36%
Mean time(s)	0,00100	0,00100	0,00100	0,00100
Longest time(s)	0,01806	0,01515	0,01402	0,00914

The Kolmogorow-Smirnov tests indicates that the results are not normally distributed for any dataset. The amount of motors being simulated has no significant effect on the system tick. Figure 9.6 shows the system loop for 1000 samples measured with a 16 motor hydraulic simulation running. This figure shows an interesting pattern: After about 250 packets, periodically, single packets are delayed significantly. This patterns is observable in all results obtained in this test.

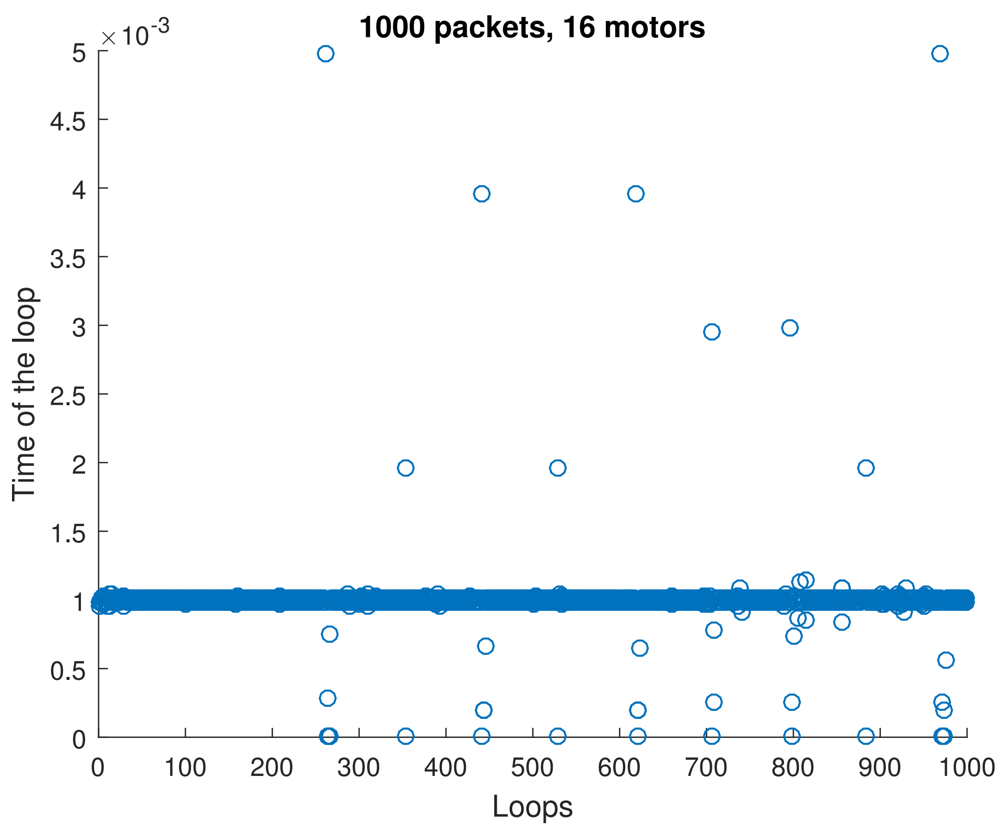


Figure 9.6: 16 motor hydraulic simulation setup. A total of 1000 samples are shown with the system loop time.

9.6.4. Interpretation

From the test results the following can be interpreted:

- The data flow between each component in the HSU is implemented correctly.
- The 1ms tick is not always achieved. After around 250 ticks a disruption starts which repeats itself every 100 packets is observed. The introduced delay is anywhere between 2ms and 12 ms. A possible explanation is that the Linux OS switches to a different thread. Further investigation is necessary to confirm this.
- The mean system tick is exactly 1ms
- When a tick takes too much time this is compensated in the next iteration. This keeps the simulation time synchronized with the HMC05 time.
- Running a heavier simulation (ie. 16 motors) may increase simulation time but does not influence the system tick significantly, this is conform the results found earlier which concluded that the simulation is fast enough till 40 motors.

10 Discussion

Although the simulation system cannot be tested as a whole, it is possible to validate the implementation by interpreting the test results. Table 10 shows the status each sub module of the HSU.

Component	Design	Implemented	Module tested
Ethernet	✓	✓	✗
Data management & IO interpreter	✓	✓	✓
Simulation	✓	✓	✓*
Monitoring	✓	✗	✗

10.1. Ethernet module

The Ethernet connection has been designed in collaboration with the Ethernet group [4]. The communication protocol is implemented throughout the system. The implementation has been tested in Test 9.2. It is however found that the strict tick of 1ms is not always met. This has to be improved in order to meet the requirements.

10.2. Data structurer and IO interpreter

The data structuring manager and IO interpreter are designed and implemented. This module complies with the requirement that Huisman commissioning engineer must be able to configure the system. These components are tested using a setup involving a Linux computer with a Python script which is described in Chapter 9.3. The results were promising as the system scaled to at least 150 unique components.

10.3. Simulation

The dynamic behaviour of hydraulic and mechanical machinery has been modelled and implemented within a modular simulation framework. The simulation setup can be configured by a Huisman engineer and a HedoN engineer can easily expand the simulation by adding additional simulation blocks. The simulation engine has been tested with a full simulation system, with a number of input sequences and has been validated by comparison to real world measurements of hydraulic plants. This test can be found in Section 9.4. It has not been tested whether this simulation is accepted by the HMC.

10.4. Monitoring System

A monitoring system has been designed and has been integrated in the overall system architecture. The monitoring system has not yet been implemented, however it has been designed to meet all requirements.

10.5. System requirements review

The system requirements were based on the system use cases. These requirements demand flexible and modular reconfigurable components. The HSU submodules have been designed as such, however the use cases have not been validated with Huisman commissioning engineers as the overall system is not yet completed.

11 Conclusion

11.1. General conclusions

The goal of this thesis is to create a testbed that can simulate the electric I/O of hydraulic and mechanical machinery for the HMC. The proposed simulation system architecture is designed to be modular and flexibile, such that the system is highly configurable and can also easily be used for other hardware in the loop simulations.

A communication protocol between the IO hardware components and the HSU master has been designed and implemented using a low level Ethernet. A generic pindriver has also been designed to emulate a large number of I/O pins.

This thesis focused on the dynamics involed in the hydraulic and mechanical simulation and the software framework within the HSU master. A simulation framework has been designed that can be configured to simulate all winch-based hydraulic setups. The framework allows for easy expansion to higher order models. A data structuring module is designed to optimize the packing and unpacking of Ethernet packets and an IO interpreter module is designed to translate analog measurements to context rich simulation variables. A monitoring module has also been designed to visualize the simulation to the user and to allow the user to simulate a sudden cable break.

Although the simulation system has never been connected to the HMC, the general architecture of the simulation system has been tested and the implementation of the sub components has been verified through a number of tests. The test results show that the current simulation system is not yet real time, as the 1 ms system tick is not always observed.

11.2. Future work and recommendations

Based on the experience gained during this 10 week project the following recommendation are made:

- Based on the test results it likely that the Linux kernel interrupts the 1ms loop every once in a while which disrupts the strict 1ms system tick. A solution to this could be to run a Real time Linux OS (such as RTOI) or to switch back to bare-metal programming. Alternatively, this problem could be solved by using a faster microprocessor or by switching to a multicore processor and by running the simulation in parallel with the communication.

- The simulation is written around an intelligent data structuring management system. The simulation consist of software objects which simulate components in a hydraulic system. Simulink is a simulation platform that provides the possibility to export an implementation in C++. It may be possible to integrate that implementation with the existing data structurer, IO interpreter and communication module. This would allow for the simulation system to be designed on the matlab/Simulink platform.

- The system is designed to work without an monitoring system, it may be a good thing to integrate a computer in the simulation system. This can be for example a raspberry pi hosting a web server. The advantage of this design is that no proprietary software must be running on the external monitoring computer except for a browser. An additional CPU is needed as the HSU cannot run the webserver and database without voiding the realtimeness.

- The physical implementation of the HSU in industry server rack hasn't be designed.
- If the HSU would be a implemented on a multi-core processor it could be possible to run a webpage on the HSU itself.

A
Glossary

The following are used multiple in this thesis:

HMC05: The fith version of a secondary hydraulic controller for sea cranes developed by HedoN for Huisman

I/O Hardware Module or Hardware I/O components. Electronic components that are connected to an Ethernet switch, form the bridge between digital signals and the analog signals of the HMC05. Not to be confused the IO interpreter (which is a software object).

HSU: Hydraulic Simulation Unit, this is implemented on the SAMAD3 chip. Its main task is to perform the hydraulic simulation. It can via communicate with IO hardware modules and a monitoring system.

IO interpreter (modules): A software layer which translates currents and voltages to values with context such as pressure and flow. Not to be confused with I/O hardware module, which is an electric component.

Simulation System or Hydraulic Simulation System: The complete system as in figure 1.1 without the HMC05 and the external PC.

Hardware in the Loop or HIL: The simulation of a plant in order to test how well a controller can control this plant.

LVDT: Linear Variable Differential Transformer, a sensor used to measure lineair displacement

Servo: 4 way hydraulic directional control valves

Encoder: A device which converts angular posistion to a analog pulse

Bibliography

[1] *Der Hydraulik Trainer*. Mannesmann Rexroth, 1986.

[2] austinmarton. Receive a raw ethernet frame in linux. `https://gist.github.com/austinmarton/2862515`, 2012.

[3] austinmarton. Send a raw ethernet frame in linux. `https://gist.github.com/austinmarton/1922600`, 2012.

[4] S. de Jong and J. Kerkhof. Communication for hydraulic simulation purposes. bsc thesis, June 2017.

[5] T.J. de Smalen and R.V. Prins. A wide range input and output driver for hydraulic simulation system. bsc thesis, June 2017.

[6] Atmel Inc. Atmel sama5d3 xplained. http://www.atmel.com/tools/ATSAMA5D3-XPLD.aspx, 2014. Accessed: 01-05-2017.

[7] Jack Johnson. Hydraulic-electric analogies: Hydraulic power conversion, part 3, Jan 2016. URL `http://www.hydraulicspneumatics.com/hydraulic-pumps-motors/hydraulic-electric-analogies-hydraulic-power-conversion-part-3`.

[8] Toyohisa Kaneko and Bede Liu. On local roundoff errors in floating-point arithmetic. *J. ACM*, 20(3):391–398, July 1973. ISSN 0004-5411. doi: 10.1145/321765.321771. URL `http://doi.acm.org/10.1145/321765.321771`.

[9] Michael Kerrisk. *The Linux Programming Interface: a Linux and UNIX system programming handbook*. No Starch Press, 2010.

[10] M.J. Lopez L. Garcia and J. Lorenzo. Hardware-in-the-loop environment for control systems evaluation under linux/rtai. In *6th WSEAS International Conference on Applied Computer Science, Tenerife,*, volume 69, pages 7499–7506, 2006.

[11] Herbert E. Merritt. *Hydraulic control systems*. J. Wiley, 1967. ISBN 978-0471596172.

[12] Van Mieghem P. *Data Communications Networking*. Techne Rress, Delft, Netherlands, 2014.

[13] Karl Pettersson. Secondary Controlled Swing Drive. Master's thesis, Linkoping University, institute of technology, Sweden, 2009.

[14] Herman Rave and Mathijs Weskin. *Hydraulieksimulatie - Opdrachtomschrijving*. HedoN Electronic developments, 2017.

[15] *Secondary control with A4VSO axial piston units*. Rexroth Bosch Group, 12 2012. re92057.

[16] W. Shen and J. Jiang. Analysis and development of the hydraulic secondary regulation system based on the cpr. In *Proceedings of 2011 International Conference on Fluid Power and Mechatronics*, pages 117–122, Aug 2011. doi: 10.1109/FPM.2011.6045741.

[17] W. J. THAYER. Transfer functions for moog servovavles. Technical Bulletin 103, 1 1965.

[18] Koen van Vliet and Herman Rave. *HMC05 Hydrauliek Simulator Systeemontwerp*. HedoN Electronic developments BV, 2017.

[19] Wikipedia. Euler method, 2017. URL `https://en.wikipedia.org/wiki/Euler_method`.