UiA University of Agder

Master

Development of a HIL Simulator for Hydraulically Actuated Cranes Using TwinCAT and Unreal Engine

JOAKIM DAHL WOLD

SUPERVISOR
Daniel Hagen, Phd
Teodor Aune, MSc
Harald Sangvik, MSc

University of Agder, 2022
Faculty of Engineering and Science
Department of Engineering Sciences

Obligatorisk gruppeerklæring

Den enkelte student er selv ansvarlig for å sette seg inn i hva som er lovlige hjelpemidler, retningslinjer for bruk av disse og regler om kildebruk. Erklæringen skal bevisstgjøre studentene på deres ansvar og hvilke konsekvenser fusk kan medføre. Manglende erklæring fritar ikke studentene fra sitt ansvar.

1.	Vi erklærer herved at vår besvarelse er vårt eget arbeid, og at vi ikke har brukt andre kilder eller har mottatt annen hjelp enn det som er nevnt i besvarelsen.	Ja
2.	**Vi erklærer videre at denne besvarelsen:** • Ikke har vært brukt til annen eksamen ved annen avdeling/universitet/høgskole innenlands eller utenlands. • Ikke refererer til andres arbeid uten at det er oppgitt. • Ikke refererer til eget tidligere arbeid uten at det er oppgitt. • Har alle referansene oppgitt i litteraturlisten. • Ikke er en kopi, duplikat eller avskrift av andres arbeid eller besvarelse.	Ja
3.	Vi er kjent med at brudd på ovennevnte er å betrakte som fusk og kan medføre annullering av eksamen og utestengelse fra universiteter og høgskoler i Norge, jf. Universitets- og høgskoleloven §§4-7 og 4-8 og Forskrift om eksamen §§ 31.	Ja
4.	Vi er kjent med at alle innleverte oppgaver kan bli plagiatkontrollert.	Ja
5.	Vi er kjent med at Universitetet i Agder vil behandle alle saker hvor det forligger mistanke om fusk etter høgskolens retningslinjer for behandling av saker om fusk.	Ja
6.	Vi har satt oss inn i regler og retningslinjer i bruk av kilder og referanser på biblioteket sine nettsider.	Ja
7.	Vi har i flertall blitt enige om at innsatsen innad i gruppen er merkbart forskjellig og ønsker dermed å vurderes individuelt. Ordinært vurderes alle deltakere i prosjektet samlet.	NA

Publiseringsavtale

Fullmakt til elektronisk publisering av oppgaven Forfatter(ne) har opphavsrett til oppgaven. Det betyr blant annet enerett til å gjøre verket tilgjengelig for allmennheten (Åndsverkloven. §2).
Oppgaver som er unntatt offentlighet eller taushetsbelagt/konfidensiell vil ikke bli publisert.

Vi gir herved Universitetet i Agder en vederlagsfri rett til å gjøre oppgaven tilgjengelig for elektronisk publisering:	Ja
Er oppgaven båndlagt (konfidensiell)?	Nei
Er oppgaven unntatt offentlighet?	Nei

Acknowledgements

I would like to thank my supervisor Daniel Hagen, and Ocean Infinity Marine previously known as Red Rock Marine for giving me the opportunity to work with such an exciting master thesis. They have been most welcoming, and throughout my time there, they have supported me from start to finish, and even provided me with an office space where I could work. They have also supplied me with CAD files, system specifications, and hardware for running the system. I would also like to thank my co-supervisors, Teodor N. Aune, and Harald Sangvik at Ocean Infinity Marine. Their continuous support and will to teach and advice me who had no prior experience in using either TwinCAT for PLC software development or Unreal Game Engine, have been of greatly appreciated. I can safely say that working with this master thesis have provided me with a set of skills and insight, that will be beneficial to me in my career as an engineer within Mechatronics. It has also made me appreciate sleep a lot more, as many late nights were used to program and solve errors within the system. Through the last six months, this have been my all and everything, and programming PLC and running simulations have even become part of my life during this last semester as i have been employed to work with PLC systems. Lastly I would like to give special thanks my partner at home, who helped me stay positive when errors occurred while providing me with sufficient time to work on my masters.

Joakim Dahl Wold, 26.05.2022

Abstract

Realistic simulations of heavy-duty machinery have the potential to reduce costs before manufacturing, increase functionality, and even save lives. Therefore it is essential to develop systems that can use realistic real-time simulations to test systems extensively before applying them to a work area.
This thesis presents a real-time Hardware-In-the-Loop (HIL) simulator system for hydraulically actuated cranes. For simulating the knuckle boom crane used as a case study, Beckhoff's TwinCAT eXtended Automation environment (XAE) software running on a PC is used as the HIL simulator platform. At the same time, Unreal Engine running on the same PC takes care of dynamics computation using inputs from TwinCAT to simulate motion by using the built-in physics engine PhysX. Components representing the electro-hydraulic actuation system are modeled through well-known equations and implemented in TwinCAT using the object-oriented Structured Text PLC programming language and can be adapted to different crane configurations by changing parameters. Unreal Engine is set up to calculate dynamic movement programmed in Unreal Engine's own Blueprint visual scripting language. Using a modded version of Beckhoff's ADS communication protocol, communication between the TwinCAT and Unreal Engine is enabled.

To validate the proposed HIL simulator, a Simulink multi-body simulation model connected to subsystems represented by MATLAB function blocks, including the same equations used in TwinCAT to represent the actuation system, is used for comparison. The knuckle-boom crane is actuated by opening and closing the electro actuated directional control valve, which introduces flow and pressure build-up in the system, resulting in a piston force that is provided as input (i.e., forward dynamics) to the multi-body system, representing the mechanical system.
Two tests are carried out to investigate the performance of the HIL simulator. First, to verify the real-time performance of the electro-hydraulic simulator running in TwinCAT, a test including only TwinCAT with a simplified mechanical representation of the actuator motion is presented. Secondly, a full-scale test, including the Unreal Engine, is presented to evaluate the performance of the co-simulation realized through the ADS communication.
The results from the testing show that while TwinCAT can be used for real-time simulation for hydraulic and mechanical computation, connecting the system to Unreal Engine for dynamic simulation introduces much room for error. This is mainly caused by Unreal Engine‘s handling of physics tied to the variable frame rate. A force that should be adequate for the movement of mechanical parts, run using Unreal Engine on a high-end computer, may only introduce slight movement, depending on the computer running the game engine.

Contents

List of Figures

List of Tables

Abbreviations

These are abbreviations used for writing the thesis:

HMI	Human Machine Interface
IEC	International Electrotechnical Commission
PLC	Programmable Logic Controller
GVL	Global Variable List
ADS	Automation Device Specification
repo	Git Repository
IPC	Industrial PC
HIL	Hardware-In-the-Loop
TF	Transfer Function
FB	Function Block
STRUCT	Structure datatype
Mesh	Structural build of 3D model

Nomenclature

β	Bulk modulus
β_i	Effective bulk modulus
β_o	Hydraulic fluid bulk modulus
Δp	Pressure difference
$\dot{p}_A$	Pressure gradient bore side
$\dot{p}_B$	Pressure gradient annulus side
ω_{BW}	Bandwidth frequency
A	Bore end surface area
a	Acceleration
a	Annulus surface area
C_x	Damping coefficient
F_C	Coloumb friction force
F_f	Static friction force
F_L	Load force
F_v	Viscous friction force
F_S	Stribeck friction force
$K_f a$	Friction force approximation constant
$K_f t$	Static friction time constant
K_v	Valve flow coefficient
K_{air}	Adiabatic air constant
K_x	Spring coefficient
m_L	Load mass
n_{air}	Volumetric air constant
p_A	Pressure bore side
p_B	Pressure annulus side
p_i	Capacitance pressure
p_{atm}	Atmospheric pressure
p_{crack}	Cracking pressure
p_{in}	Incoming pressure
p_{out}	Outgoing pressure
Q	Flow rate
Q_{in}	Incoming flow rate
$Q_{nominal}$	Rated flow rate
Q_{out}	Outgoing flow rate
v	Velocity
V_{line}	Line volume
$X(s)$	Input
x	Cylinder piston position
x_{max}	Piston max position
$Y(s)$	Output
ζ	Damping ratio

Chapter 1

Introduction

1.1 Background

This master thesis focuses on the creation and steps required to build a HIL simulator of a knuckle boom crane. The project will make use of multiple types of software to simulate a complete system. Beckhoffs TwinCAT 3.1 will be used as the engineering environment (XAE), and real-time kernel (XAR), for the PLC programming part of the system, which will use structured text [7] as the coding language. Matlab/Simulink will be used for testing and validation of the finished PLC system, and used for comparison for actuation of the crane. For visualizing the system, the game engine software Unreal engine from Epic games is to be used. All the 3D models used in the game engine will be supplied from Ocean Infinity Marine, and will be used to visualize and process dynamic movement based on forces received from TwinCAT. Making the crane move in real-time with TwinCAT calculating hydraulics based on motion feedback from Unreal engine. The 3D rendered scene in the game engine will then move according to the forces being fed to the hydraulic actuators. The system setups made in Simulink and TwinCAT+Unreal Engine will then be run through some tests to check the functionality and performance of the system.

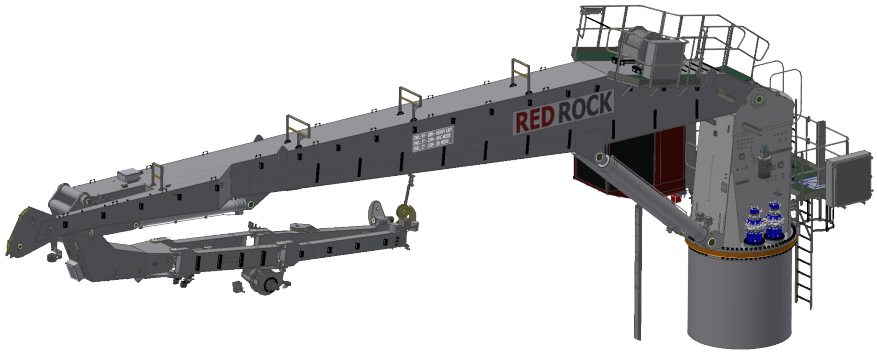


Figure 1.1: The CAD model representation of the crane

1.2 State of the Art

The type of system to be created have not been done before, where all the dynamic movements are computed in Unreal engine, while the process of actuation is happening within the PLC to create a combined control and compute system for knuckle boom cranes in real-time. However research have been done within various aspects of the systems used, which will further be used to build upon to create the real-time TwinCAT's knuckle boom crane simulation system. Previous work done by others when it comes to the following topics will be looked at for possible use, improvement and implementations: The use of game engines in combination with real-time control systems programmed with PLC language, the use and stabilization of knuckle boom cranes, ADS systems for communication, multibody systems in Unreal, and use of hydraulic actuators.

Previous research of HIL testing in combination with TwinCAT and visualization software through the use of game engines have been researched and developed in the past by Harald Sangvik [14]. During his master thesis, he used Beckhoffs TwinCAT PLC as the control system of a 3D compensated gangway, to control a complete digital twin of the gangway made in Unity game engine. While the calculations of dynamics for the gangway were done within Unity using C# scripting language. To enable communication between TwinCAT and Unity, Beckhoffs ADS communication protocol was used.

For system verification he used a multibody model of the gangway connected to a hydraulic system, created in the open-source software OpenModelica using native OpenModelica and OpenHydraulics library. Whereas his report focused on placing all the computing of dynamics within Unity game engine, and only using TwinCAT as a controller. This project will also use a combination of game engine and TwinCAT PLC, but rather than placing all dynamics, hydraulics, and mechanical systems within the game engine, the hydraulics will be computed using TwinCAT PLC as the main component of the system, sending forces over to the Unreal Engine to calculate dynamic movement and sending that information back to TwinCAT's hydraulic system.

1.3 Motivation

For companies working with engineering and manufacturing of crane systems as part of their business, it is important to able to provide their customers with short delivery times, error-free software, and reliable support. Therefore the work done within this project is of high importance, since it focuses on creating pre-programmed reusable PLC software able to control cranes of various dimensions. For creation of the reusable PLC software, object oriented programming can be used. Combining this with a visual simulation of the system done in a game engine, makes it possible to see and test each crane in a real-time environment before deploying the real crane. Using Unreal Engine which is a free game engine available to everyone, makes the system highly available for further development and support, and means that most testing can be done in-house using regular computers. While supporting customers remotely, a digital twin of the same crane they are requesting support on can be simulated which makes it easier to get a better view of the issues they are trying to solve.

If the project is successful, it will perhaps help in lower costs regarding control software and development in general for knuckle boom cranes, as it will minimize the time and resources required to create control systems for newly produced cranes. A major reason for this, is that a crane can be simulated, tested and seen, control systems for it can be created before having the physical crane ready. Along with this, the PLC simulation program will be streamlined, making it easy to use even for new employees, as there will only be a need to

adjust parameter values for arm lengths and hydraulic actuators to adapt the PLC from one crane to another.

1.4 Problem Statement

Is it possible to simulate the hydraulic actuation system of a knuckle boom crane using TwinCAT, and how does it perform in combination with the simulation of a multibody system in Unreal Engine?

To answer the research question, the following objectives will have to be completed:

1. Build a simulator of actuation systems in TwinCAT using the Structured Text PLC programming language.
2. Build a multibody simulation able to compute dynamics in Unreal Engine.
3. Make TwinCAT and Unreal Engine able to communicate.
4. Performance testing with comparison against a Simulink simulation.

1.5 Outline

Chapter 1 introduces the master thesis and describes the project. Giving insight to the overall project, background, motivation, problem statement, and research objectives.

Chapter 2 describes the theory behind the system, mainly the mathematical approach used to create the equations used to modem the system.

Chapter 3 Covers the methods used to answer the research question. This includes development of systems in TwinCAT and Unreal Engine, and the setup of a Simulink multibody system for comparing systems for validation.

Chapter 4 presents the results of the various aspects of the system, and gives comparisons to the system with and without certain functions.

Chapter 5 is used for discussing changes to the project, deviations, and also goes into areas of future development that can and should be introduced to the crane system in the event of continuing the work done so far.

Chapter 6 concludes the thesis.

Chapter 2

Theory

To create the multibody systems for Unreal Engine and Simulink, existing libraries found within the two software will be used. In Simulink, Simscape graphical code blocks will be used, while in Unreal Engine visual programming via Unreal's Blueprint coding language will be used to put together the multibody system. The hydraulic and mechanical system will be set up using the equations found in this chapter.

2.1 TwinCAT

TwinCAT is an open-source engineering environment for creating robust PLC systems. It runs using two systems, the run time kernel (XAR) and TwinCAT eXtended Automation Environment (XAE) [8]. Both of these can be installed on a standard computer at no expenses. The code for the PLC system is written via the XAE which uses Visual Studio (VS) as its development environment [13].

2.1.1 Codesys PLC Compiler

The open-source TwinCAT is based on Codesys which means that that the code written using it, is cross platform capable. This means that the code which is written in Structured Text can be copied directly to and from other Codesys based environments. Being based on Codesys also means that the programming done follows the IEC 61131 standard described in chapter 2.1.2 below.

2.1.2 International PLC Standard

For programming the PLC software in structured text, IEC 61131[7] is used as the programming Standard. This is a Standard used worldwide within automation software for programmable controllers. Following IEC 61131 means that the basics of the software architectural setup is clearly defined, and easily understood by most. To fulfill the agreements of IEC 61131 Standard, the code must be defined in the following ways: The code must be written in the most common language used in the world which is English, with datatypes such as BOOL, LREAL, INTEGER, and their value range clearly defined, the variable types such as global, local, inputs, etc must be defined, the variable types can be assigned to variable classes such as VAR, VAR_GLOBAL, VAR_INPUT etc, how the PLC is set up by using programs and tasks, and Program Organization Units such as functions, function blocks, programs etc are defined.

TwinCAT3 uses an extended version of the IEC, mainly the IEC 61131-3, which is part three of the IEC Standard. This part defines five programming languages that can be used to program a PLC, which are the following: Instruction List (IL), Function Block Diagram (FBD), Ladder diagram (LD), Structured text (ST), and Sequential Functon Chart (SFC).

This project will use Structured text as its main programming language which loosely resembles programming languages like Python and C. This is advantageous as MATLAB which uses a language based on C, will be used throughout the project, making it easier to grasp.

There are multiple advantages of programming using IEC 61131-3, the main ones are: It simplifies the understanding of code written for PLC programmers, being able to reuse written program code into other PLC software, efficiency and time saving, and with it being widely used and well known among the automation industry makes it a good choice since there are many guides on the usage both online and in books.

2.1.3 PLC Software Development

To be able to control the knuckle boom crane, Beckhoffs PLC software TwinCAT3 was used as the main component of the system. PLC programs written in TwinCAT3 has the feature of being able to be implemented onto dedicated Beckhoff IPC, and I/O cards to run the system in real-time with using a small form factor proprietary hardware. The dedicated Beckhoff IPC has the same hardware as a regular PC, where the only difference is that the components of the IPC has industrial quality components and a BIOS that is optimized better for real-time performance.

As mentioned in chapter 2.1, TwinCAT runs using its own run time kernel XAR, which makes it possible to share cores with Windows or use one dedicated to itself. This can be used as a safety measure, when using a dedicated core, if Windows crashes, the XAR keeps running on its own core.

Equations for the hydraulic system were to be set up and programmed into the PLC, in such a way that it could take variables, dimensions, and commands for hydraulic actuator systems as inputs. While outputting desired computed results like positions, velocity, and forces. These outputs are then to be sent over into a game engine software and used to visualize the movements of the digital twin made of the knuckle boom crane.

The hydraulic system was made using a well established approach [10] for setting up equations for hydraulic cylinders, orifice flows, valve dynamics, effective Bulk modulus and pressure components. These were to be made using function blocks containing named variables inside each equation, making it possible to reuse the function blocks wherever needed applying various sets of variables as required.

2.2 Electro-Hydraulic Actuation System

The models for hydraulics and mechanical were set up using well known equations, and are known to be satisfactory for simulation of similar systems [6].

2.2.1 Hydraulic Actuators

Hydraulic cylinder

To be able to simulate motion of the hydraulic cylinders, the very well known Newtons second law of motion F = ma is used. The governing equation for movement of hydraulic actuators below was set up as seen below in equation 2.1. This equation yields the acceleration of the hydraulic piston inside the cylinder, which when integrated once and twice, produces the current velocity and position of the hydraulic piston.

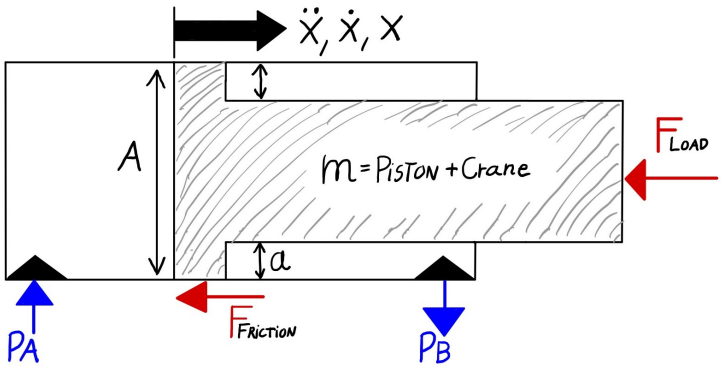


Figure 2.1: Illustration showing how the components affect an hydraulic cylinder

$$\ddot{x} = \frac{p_A \cdot A - p_B \cdot a - F_L - F_{friction}}{m_L} \tag{2.1}$$

$$\dot{x} = \int \ddot{x} \tag{2.2}$$

$$x = \int \dot{x} \tag{2.3}$$

The integrated equation 2.2 produces the velocity of the piston in [m/s], while integrating this a second time as shown in equation 2.3 produces the current Position of the piston in [m].

2.2.2 Piston Forces

An important part of any hydraulic - mechanical system is to be able to calculate the summation of forces acting upon the cylinders pistons at all times. The following equation were set up as governing equation for piston force.

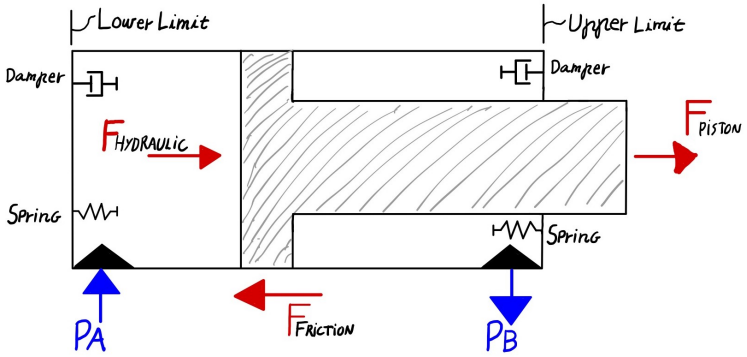


Figure 2.2: Illustration forces generated in the hydraulic cylinder

$$F_{Piston} = F_{Hydraulic} - F_{StribeckFriction} - F_{UpperStrokeLimit} + F_{LowerStrokeLimit} \tag{2.4}$$

Equation 2.5 was set up for calculating the resulting hydraulic force pushing on the piston.

$$F_{Hydraulic} = p_{in} \cdot A - p_{out} \cdot a \tag{2.5}$$

This equation uses the pressures at each side of the cylinder along with the size of areas inside of the cylinder to generate a variable hydraulic force.

To add friction to the moving pistons, equation 2.6 representing a Stribeck friction force was set up. This friction force adds to the stability of the system by providing an exponentially higher force as the pistons velocity increases. This helps make the system run more smoothly and leads to the system stabilizing quicker when the crane is signaled to stop at its current position.

$$F_S = (\exp\left(\frac{-v_{Piston \cdot tmp}}{K_{ft}}\right) \cdot F_f + F_C) \cdot tmp + v_{Piston} \cdot F_v \tag{2.6}$$

$$tmp = \exp\left(v_{Piston} \cdot K_{fa}\right) \cdot 2 \tag{2.7}$$

$$tmp = \frac{tmp - 1}{tmp + 1} \tag{2.8}$$

Where equations 2.7 and 2.8 for tmp represents a ratio that maxes out at 1. This changes depending on the current velocity of the piston. Additionally this only works in slow moving systems, and any velocity higher than approximately 0.918 [m/s] will produce errors to the system.

To stop the hydraulic cylinders pistons from breaking and moving outside of the hydraulic cylinder, equations 2.9 and 2.10 simulating upper and lower stroke limits were set up.

$$F_{USL} = (x_{Piston} - x_{Max}) \cdot K_x + C_x \cdot v_{Piston} \tag{2.9}$$

$$F_{LSL} = x_{Piston} \cdot K_x + C_x \cdot v_{Piston} \cdot (-1) \tag{2.10}$$

These two equations apply a force to stop the piston from moving outside its area of operation, using springs and dampers to bring it to steady-state. The upper stroke limit force only activate when the hydraulic piston is at or above its maximum stroke length, while the lower stroke limit only activates when the piston is at, or below zero.

2.2.3 Hydraulic Pressures

Differential equations yielding pressure gradients for each side of hydraulic actuators were set up as shown below.

$$\dot{p_A} = \frac{\beta}{V_{Line} + x \cdot A} \cdot (Q_{in} - Q_{out}) \tag{2.11}$$

$$\dot{p_B} = \frac{\beta}{V_{Line} + (x_{max} - x) \cdot a} \cdot (Q_{in} - Q_{out}) \tag{2.12}$$

These two equations will be used to compute the pressure nodes found in the hydraulic actuation system through integration to yield the pressure components required to simulate the hydraulic system. The resulting pressure will then be inserted into the equation for hydraulic cylinder acceleration found in 2.1.

$$p_A = \int \dot{p_A} \tag{2.13}$$

$$p_B = \int \dot{p_B} \tag{2.14}$$

Equations 2.13 and 2.14 shows the integrated pressure gradients resulting in pressure nodes p_A and p_B both in [bar].

2.2.4 Orifice Flow

To be able to determine the flow coming in and going out of the hydraulic cylinders, the orifice equation was used. This takes care of calculating the flow through the input and output ports of the cylinders. It makes use of the flow coefficient of the valve, the current pressure total in the cylinder, and a sign function that can change between positive and negative to be able to change the direction of flow depending on the pressure nodes.

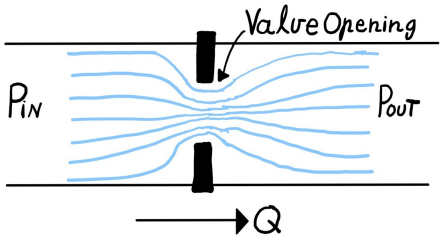


Figure 2.3: Illustration of the orifice equation, with flow running from left to right

$$Q = K_v \cdot u_{DCV} \cdot sign(\Delta p) \cdot \sqrt{|\Delta p|} \tag{2.15}$$

$$K_v = \frac{Q_{nominal}}{\sqrt{\Delta_p}} \tag{2.16}$$

$$\Delta p = (p_{in} - p_{out} - p_{crack}) \tag{2.17}$$

While equation 2.15 takes care of the fluid flow in the system, equation 2.16 concerns the valve flow coefficient and 2.17 gives the current total pressure. The total pressure uses results from equations 2.13 and 2.14, assigned as p_{in} and p_{out}, which of them is regarded as inlet and outlet depends on which way the system is currently running, extracting or retracting the hydraulic piston.

2.2.5 Directional Control Valve

For the system to be able to run both ways, a variable 4/3 valve was chosen. This valve has four connections to hydraulic flow, and can control the direction of the flow found in equation 2.15. It also has the feature of being able to gradually open and close, without limiting the options to fully open or fully closed, making the system able to partially restrict the amount of flow happening in the system.

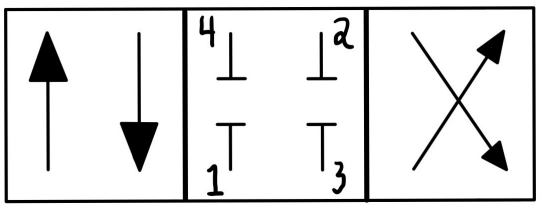


Figure 2.4: Illustration of the 4/3 Directional Control Valve

The opening of the valve is to be set a value ranging from -1 to 1, which is used to control the direction of flow in the system. This value could have been set as a static number used as a step function, but since the system is to be used to simulate a real world scenario, valve dynamics are introduced.

For the valve opening to be able to open and close in a more realistic way, a second order transfer function is used to smooth out the opening and closing of the valve. This makes the valve function ramp up and down instead of unrealistically jumping from 0 to a set value in an instant.

$$\frac{Y(s)}{X(s)} = \frac{\omega_{BW}^2}{s^2 + 2 \cdot \zeta \cdot \omega_{BW} \cdot s + \omega_{BW}^2} \tag{2.18}$$

To later implement this into the PLC program using structured text, the transfer function above had to be converted into a discrete differential equation. This was achieved through the use of an inverse Laplace transformation.
Simplifying the TF:

$$\frac{Y(s)}{X(s)} = \frac{1}{\frac{1}{{\omega_{BW}}^2} \cdot s^2 + 2 \cdot \frac{\zeta}{\omega_{BW}} \cdot s + 1} \tag{2.19}$$

$$a = \frac{1}{{\omega_{BW}}^2} \qquad b = \frac{\zeta}{\omega_{BW}} \tag{2.20}$$

$$\frac{Y(s)}{X(s)} = \frac{1}{a \cdot s^2 + 2 \cdot b \cdot s + 1} \tag{2.21}$$

$$\Rightarrow X = Y \cdot (a \cdot s^2 + 2 \cdot b \cdot s + 1) \tag{2.22}$$

$$\Rightarrow X = a \cdot s^2 \cdot Y + 2 \cdot b \cdot s \cdot Y + Y) \tag{2.23}$$

Applying inverse Laplace:

$$x = a \cdot \ddot{y} + 2 \cdot b \cdot \dot{y} + y \tag{2.24}$$

$$\Rightarrow \ddot{y} = \frac{x - 2 \cdot b \cdot \dot{y} - y}{a} \tag{2.25}$$

$$\dot{y} = \int \ddot{y} \qquad y = \int \dot{y} \tag{2.26}$$

The theory of the valve dynamic is based on a paper by Morten Bak [2], where he did experimental frequency analyzes of a Danfoss PVG32 valve and found the average values for bandwidth frequency ω_{BW} and damping ratio ζ. Specifically $\omega = 30$ [rad/s] and $\zeta = 0.8$. As these values were found to be optimal in Morten‘s paper, they will be used as function parameters for the dynamics of the valve movement in this thesis. The twice integrated result of the equation for acceleration of the valve opening 2.25, will be used to determine the current opening position of the valve.

2.2.6 Integration Method

The differential equations are integrated using forward Euler to find their integrated counterparts. Forward Euler is an approximation method to find the integral value using the current state and time increments to calculate the next state.

$$x_{n+1} = x_n + \dot{x}_n \cdot dt \tag{2.27}$$

Chapter 3

Methods

This chapter will take care describing the steps taken in order to build the system. It will give information on how the TwinCAT PLC system is built regarding both hydraulics and mechanical. The visual system setup will be explained in detail, and the communication between the two systems will then be clarified. Additionally, a multibody system will be configured to use the same hydraulic system as the PLC to actuate, and to later be used for comparison.

The hydraulics and mechanical parts of the system is split into two systems, TwinCAT and Unreal Engine. TwinCAT simulates all of the hydraulics computations which produces forces summed up into piston forces for each cylinder. Unreal Engine gets the piston force from TwinCAT over ADS, and in return computes dynamic movement of the crane. The positions and velocity from Unreal Engine is then sent back to TwinCAT. These two systems together simulates the knuckle boom crane operation.

3.1 Case Study

Knuckle boom cranes are an extension of the single-boom crane. Adding a knuckle jib to the outer part of the boom of a crane, increases its flexibility resulting in a more complex and larger workspace available for the crane‘s tool tip. This is most likely why they are becoming one of the most common types of cranes used within a variety of industrial areas [1]. The knuckle jib of the crane makes it able to fold into itself, and makes it a good tool for moving payloads from point-to-point.

Comparing the knuckle boom crane to the typical telescopic crane that only extracts and retracts, it should be noted that one of the disadvantages of using the knuckle boom is that it is more susceptible to damage. Components such as the hydraulic lines are often on the outside of the crane arms making them vulnerable, while a telescopic crane normally has these hidden away on the inside the crane arm [4].

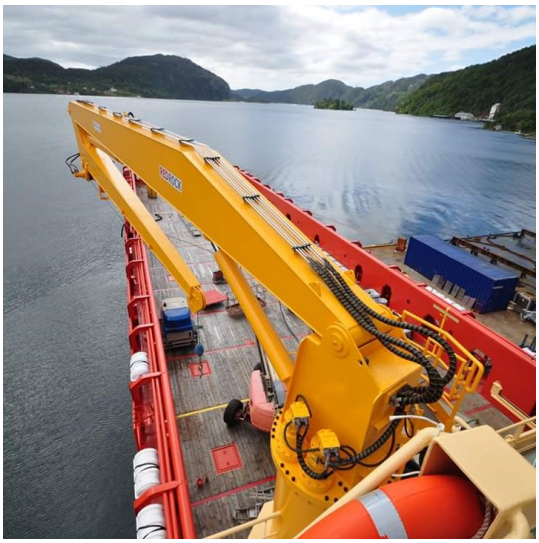


Figure 3.1: An Ocean Infinity knuckle boom crane [11]

Figure 3.2: An Ocean Infinity telescopic crane [12]

Another difference in the use of these two cranes are that while the knuckle crane is good at moving point-to-point, it is not great at placement of payloads behind obstacles, as its knuckle can accidentally come in contact with and damage said obstacle when unfolding to lower the load.

An important part of what makes the use of knuckle boom cranes so desirable for payload operations, is that they are far easier to control. This is because of the payload being placed in the immediate close proximity of the tool point eliminating instability. On a telescopic crane, the payload hangs onto a long cable that is used to lower and lift the payload, making it an unstable lifting solution in theory.

The knuckle boom crane even has multiple applications since the tool point is able to be fitted with an increasingly wide roster of attachments targeted at different purposes. These can be everything from claws for grabbing, to digging buckets, and many more.

Because of its wide variety of applications, and its flexibility when it comes to movement and reaching difficult areas, the knuckle boom is the chosen crane this thesis. Using a knuckle boom crane is a lot more complex when it comes to making the mechanical model for the system, but the benefits of the crane if implemented in time, out weights the cons. A crane fit for the future.

3.1.1 Considered Knuckle Boom Crane

As mentioned why in section 3.1, a KnuckleBoom crane is chosen to be modeled and simulated throughout the project. To do this efficiently, a study of the crane and its components were done. This was to get an overview of the amount and types of actuators required to program into TwinCAT for the crane to be able to mimic the real physical crane.

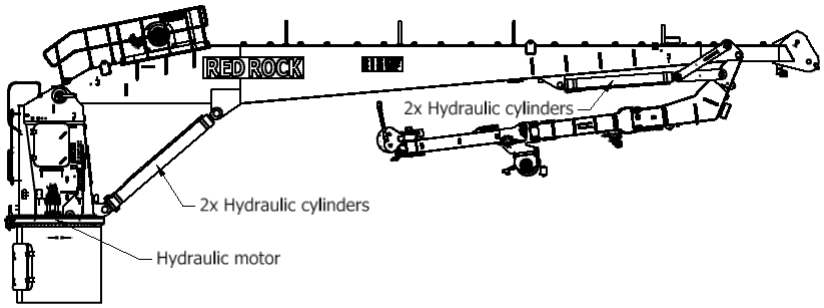


Figure 3.3: Drawing of the arm made in Autodesk Inventor, with annotations for actuators

By inspecting the drawing above, it is seen that the system requires two hydraulic cylinders for the MainBoom and two hydraulic cylinders for the Knuckle boom, as actuators for the crane arms. There is also a need for a hydraulic motor to actuate and rotate the crane‘s slew joint to be able to turn the whole crane around.

By looking at the specifications data sheet for the crane, the pistons lifting the main boom should be able to extend up to 3070mm, while the pistons lifting the knuckle jib should be able to extend to 2320mm, for the crane to reach its max position. A run time of 90s for the real crane to lift up from a starting point at rest to its max position is also noted, as this can be a sign to look for that indicates if the digital twin is behaving as close to the real crane as possible.

3.1.2 System Identification

To create the digital crane as close as possible to the real crane, the mass properties of the main components of the crane were studied using Inventor. Parameters for inertia, mass, and center of mass for the king, main boom, knuckle jib, and hydraulic cylinders were found. The values for each of these are noted in tables 3.1, 3.2, and 3.3.

King			**Boom**			**Knuckle**		
m	9548.932	[kg]	m	12778.219	[kg]	m	5344.619	[kg]
$m_{x,◔}$	0.01902	[m]	$m_{x,◔}$	-8.04374	[m]	$m_{x,◔}$	-0.09160	[m]
$m_{y,◔}$	-0.02866	[m]	$m_{y,◔}$	0.86496	[m]	$m_{y,◔}$	1.33344	[m]
$m_{z,◔}$	-1.02348	[m]	$m_{z,◔}$	0.09635	[m]	$m_{z,◔}$	4.29518	[m]
I_{xx}	15537.579721658	[kgm^2]	I_{xx}	7114.72560	[kgm^2]	I_{xx}	38034.92421	[kgm^2]
I_{yy}	14993.168777119	[kgm^2]	I_{yy}	437213.61585	[kgm^2]	I_{yy}	37329.15993	[kgm^2]
I_{zz}	7026.716838393	[kgm^2]	I_{zz}	438926.98425	[kgm^2]	I_{zz}	1911.37793	[kgm^2]

Table 3.1: Mass, CoM, and inertia of crane arm bodies

The origins set for the boom and knuckle are at the center of the revolute joints connecting them together. The king has its origin set at the center of It‘s circular bottom.

Cylinder	(Boom)		**Cylinder**	(knuckle)	
m	93.301	[kg]	m	47.854	[kg]
$m_{x,◔}$	1.44936	[m]	$m_{x,◔}$	1.14441	[m]
$m_{y,◔}$	-0.00029	[m]	$m_{y,◔}$	0.00057	[m]
$m_{z,◔}$	0.00189	[m]	$m_{z,◔}$	0.00651	[m]
I_{xx}	1.9472911	[kgm^2]	I_{xx}	0.62964	[kgm^2]
I_{yy}	134.27063	[kgm^2]	I_{yy}	46.22579	[kgm^2]
I_{zz}	134.14240	[kgm^2]	I_{zz}	46.15761	[kgm^2]

Table 3.2: Mass, CoM, and inertia of cylinder bodies

Piston		(Boom)	**Piston**		(knuckle)
m	99.933	[kg]	m	50.313	[kg]
$m_{x,◔}$	176.678	[m]	$m_{x,◔}$	141.464	[m]
$m_{y,◔}$	0	[m]	$m_{y,◔}$	0	[m]
$m_{z,◔}$	0	[m]	$m_{z,◔}$	0	[m]
I_{xx}	0.47860	[kgm^2]	I_{xx}	0.15094	[kgm^2]
I_{yy}	125.87937	[kgm^2]	I_{yy}	42.45844	[kgm^2]
I_{zz}	125.83972	[kgm^2]	I_{zz}	42.44596	[kgm^2]

Table 3.3: Mass, CoM, and inertia of piston bodies

The origins set for of the hydraulic actuators are the same for both the cylinders and the piston, and are located at the center of the bottom of each piston.

Boom actuator			**Knuckle actuator**		
Bore diameter	0.28	[m]	Bore diameter	0.22	[m]
Rod diameter	0.18	[m]	Rod diameter	0.14	[m]
Stroke length	3.07	[m]	Stroke length	2.32	[m]

Table 3.4: Sizing of the hydraulic actuators

3.2 Programming and Implementation in TwinCAT

3.2.1 PLC Software

TwinCAT3 is used to set up and program both hydraulic and mechanical models. The PLC code written uses a combination of function blocks, global variable lists, programs, functions, and STRUCTS, all written in structured text programming language.

There are multiple smaller function blocks containing code for each of the equations found in chapter 2 Theory, with the main components of the system being the function blocks "ValveControlledCylinder" and "CylinderMech". These function blocks incorporates all the other smaller blocks by calling on them to combine them all into a complete system. Examples of this is shown in Appendix BB

The setup of the hydraulic and mechanical system found within TwinCAT PLC is further explained through flowcharts in chapter 3.2.2 below.

3.2.2 Electro-Hydraulic Actuation System Model

Below is a representation of how the hydraulics part of the system in TwinCAT works, this is a simplification of the programming code setup for the system.

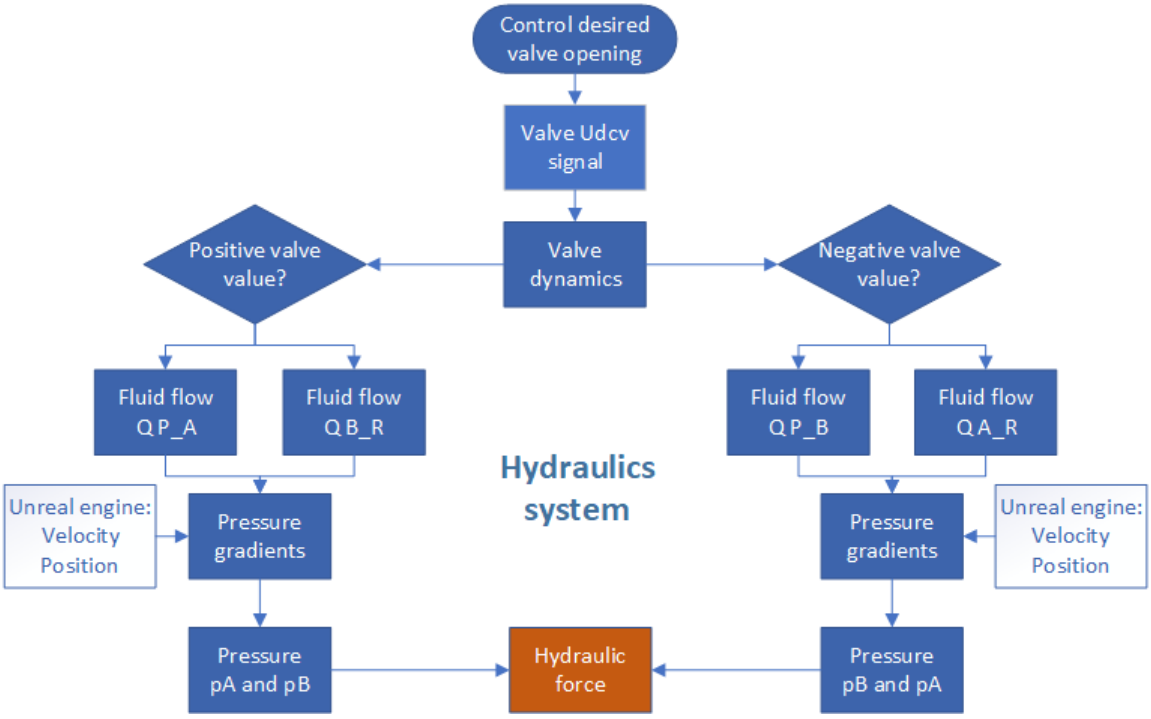


Figure 3.4: Flowchart of the PLC‘s hydraulics system

The hydraulic system is controlled by opening and closing valves for each actuator, which in return makes the system calculate the hydraulic force needed to move the piston in the desired direction. The system uses positions and velocities retrieved from hydraulic cylinders created in a multibody system in Unreal Engine. The Unreal Engine model is explained further down in chapter 3.3.1

3.2.3 Mechanical System Model

The mechanical model of the PLC consists of the mechanical aspects such as the piston force, the current velocity, and the position of the pistons. This was initially used without a connection to Unreal Engine, to be able to test the hydraulic systems functionality before moving on into further development and adding Unreal Engine as a part of the system.

The current system uses velocity and position data of the pistons retrieved from the visual Unreal Engine 3D model simulating the crane moving via dynamics coded into Unreal.

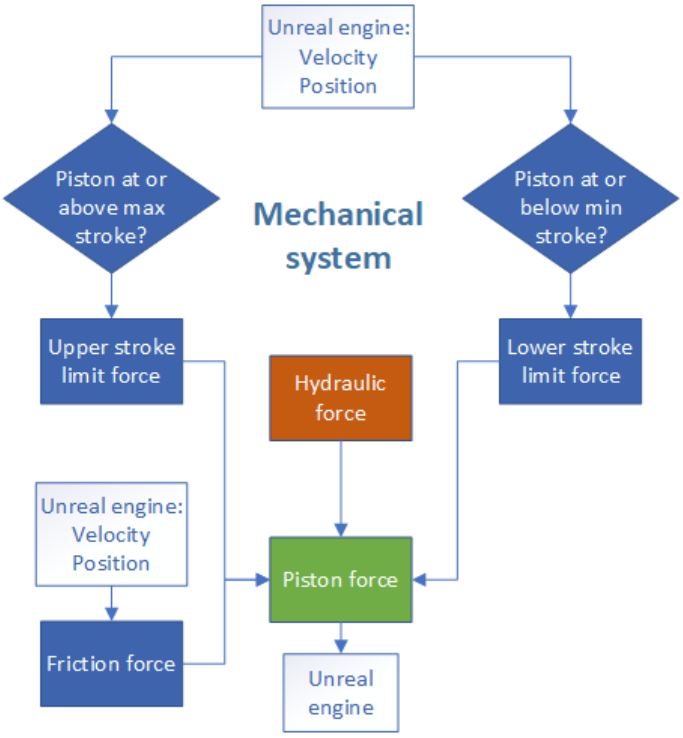


Figure 3.5: Flowchart of the PLC‘s mechanical system

The system uses the hydraulic force computed within the hydraulics actuator system, in combination with friction force and stroke limit forces. The resulting piston force is then sent to the pistons in Unreal Engine to actuate the piston movement making the crane arms move accordingly.

3.2.4 Parameters

Each component of the hydraulic system requires some parameter presets for the system to be able to function. These parameters include sizing of the hydraulic system, information about the fluid used, supply and tank return pressure, etc. The assigning of parameters to the instance of actuators can be seen in Appendix C.B.14.

By assigning the variables as persistent in the parameter block setup, each of the hydraulic actuators can be set to use different values from each other by running the system and then manually changing parameters stored for each cylinder. The affix also makes it so that the system remember the last set parameters, so there is only a need to set the parameters once. This eliminates the need of reassigning values each time the system runs, which would be the case without this function when having only one parameter file.

The standard parameters that were assigned to all actuators can be seen in Appendix B.B.16. Most of the values from the parameter code were used by the whole system, but some were changed due to the difference in sizing for each actuator. The values that required changing for each separate cylinder can be found in table 3.1.2

3.2.5 Global Variable List

For ease of use when it comes to storing variables and be able to use them by the system wherever they are needed, GVL‘s were made. These make it possible for the PLC system to temporary store values calculated by both TwinCAT and Unreal Engine, making them available for available global use. Separate GVL´s for variables received and variables sent

between TwinCAT and Unreal Engine were made as to have complete control of the communicated variables. By storing them in this way, values for force, positions and velocities can be temporary stored and then be used in function blocks in the program that uses these values.

3.2.6 Cycle Time

To help with the stability of the system, the cycle time was set fairly low at 100μs. As the cycle time is lowered, the program increases its frequency of calculations using smaller increments regarding time calculations, which leads to the system being more accurate and stable. The main thing to think about when adjusting this value is that lowering cycle time also requires more computing power, so finding a comfortable spot where the system is running fast enough, while still being stable is highly focused on. A cycle time that is too high may also lead to the program reaching singularities where there are none, which would crash the system. To match the cycle time set, the timer that was coded into the the main program program that initializes the PLC, was also set to use the same time increments.

3.2.7 Software Re-usability

As mentioned earlier, the PLC written in TwinCAT was made with the intention of being able to reuse all the different components whenever needed. For this an object oriented programming approach was used, meaning essentially making programmed simulated versions of components found in physical hydraulic, and mechanical systems.

All functions added to the system was made using named variables and interchangeable parameters in their calculations instead of hard coding numbers into the various equations. This meant that they can be reused for different purposes by redefining the inputs sent to the said function block.

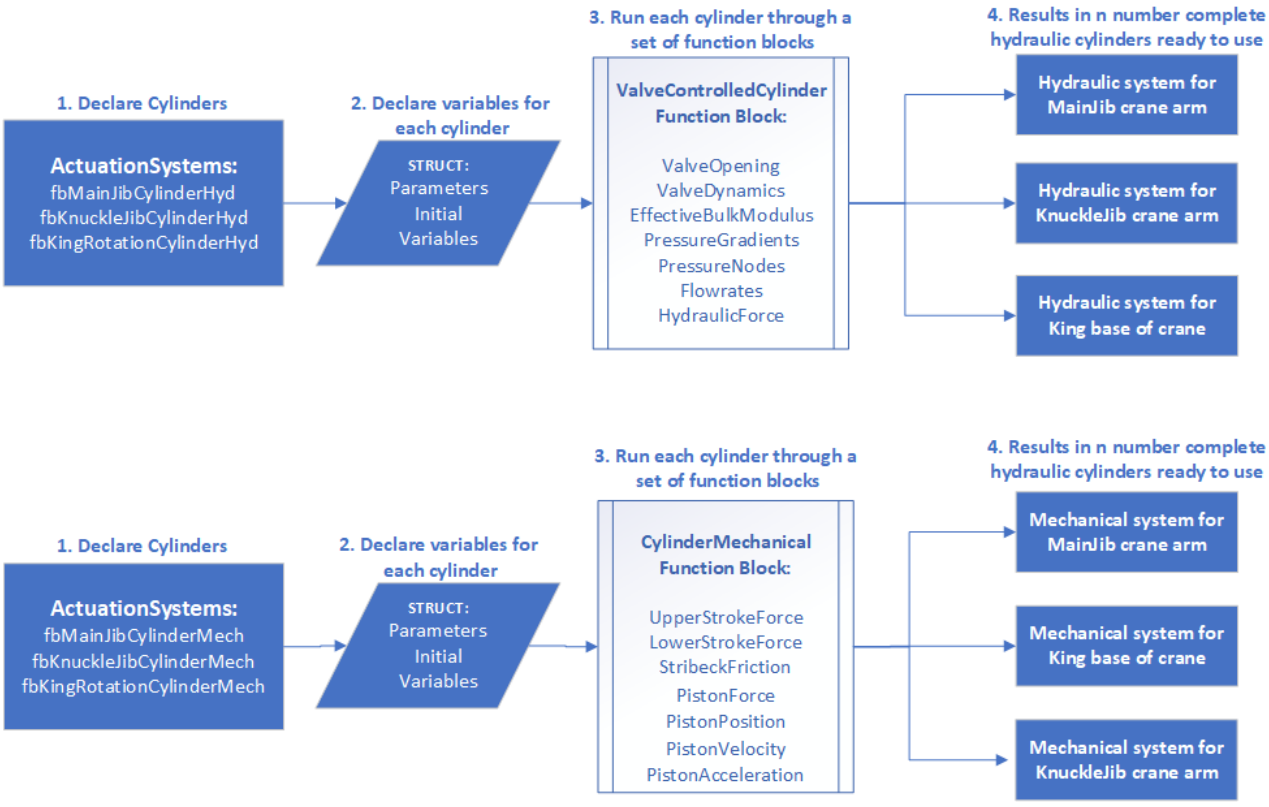


Figure 3.6: Flowchart showing how the system creates multiple hydraulic-mechanical cylinders

The picture above shows how the function blocks were reused multiple times to be able to generate hydraulic systems for different parts of the crane reusing the same function blocks.

3.2.8 HMI

To monitor and control the hydraulic cylinders in TwinCAT, an HMI overview was made. This was composed of variables for current position of the pistons, and the opening of the valves. The HMI also includes buttons that enable or disable direct control of the valves via the HMI. The valve opening can be set either by pulling the sliders to the sides or clicking the variable field U_{DCV} and entering a number manually.

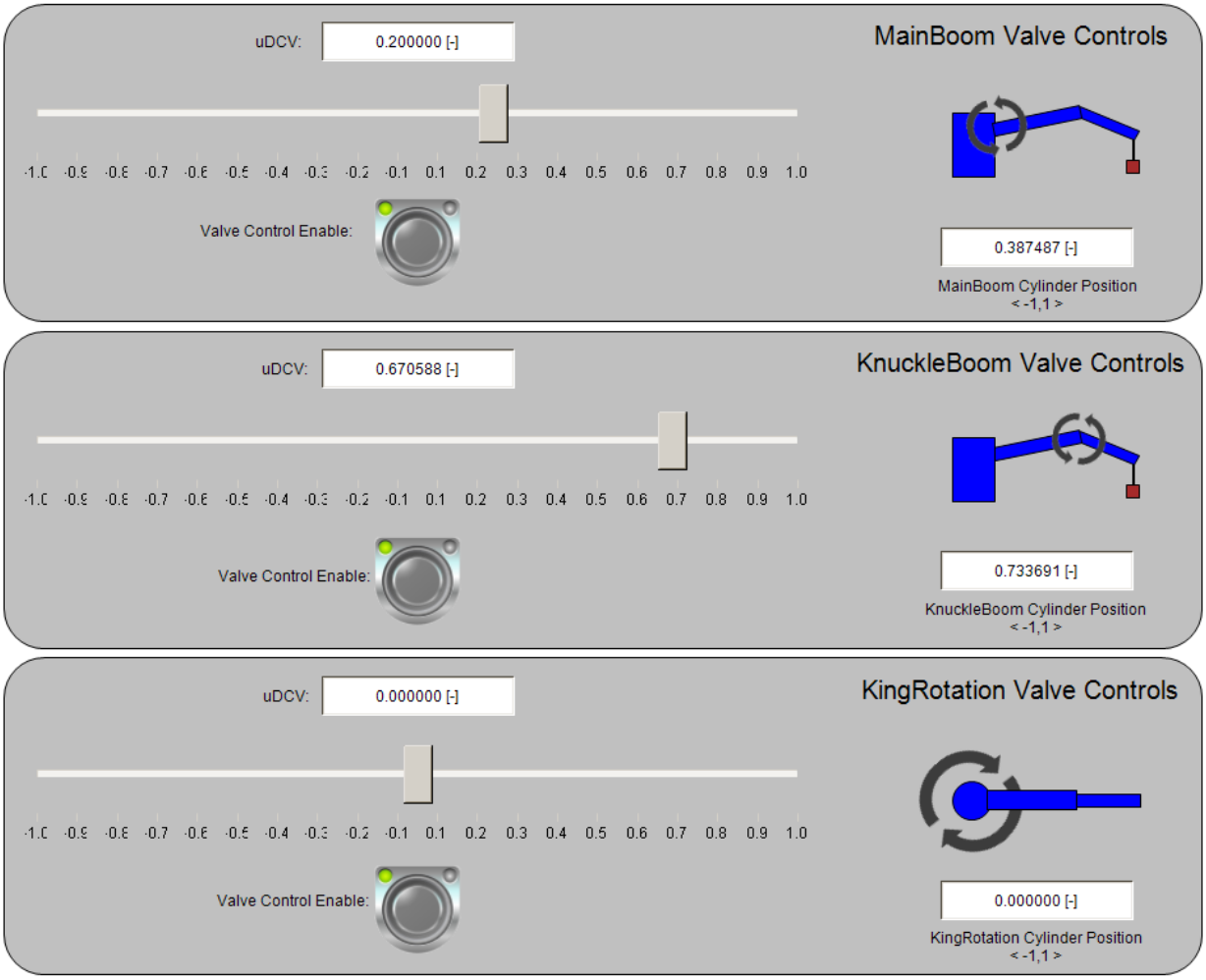


Figure 3.7: TwinCAT HMI displaying valve openings and cylinder positions

Figure 3.7 above shows the HMI as the system is running, where the valve of the MainBoom is at a positive 20% opening with its piston currently extracted 0.39 [m]. If the opening value for the valves are set to a negative value, the cylinders pistons would start to retract instead.

Remote Control

To add external control there would be a need for a physical device. It was decided to implement communication between an Xbox controller and the TwinCAT PLC program. To do this a git repository from Github [3] was cloned into the existing TwinCAT folders. This repo contains python code which is usable by first starting the TwinCAT PLC and then running the included python script. Using this enables valve operation through the use of the Xbox controller‘s joysticks, resulting in a change of flow rates and thereby extracting

or retracting the cylinders pistons. This needs to be run on the same IPC as TwinCAT to function.

Figure 3.8: Controller system overview

3.2.9 Latency

The system was to run as close to real-time as possible, and therefore the latency within TwinCAT had to be looked at. Beckhoff's TwinCAT PLC software is greatly integrated into windows and is able to both share core usage with windows, or use one or more cores exclusively. Through the PLC's Real-Time interface window, core settings can be configured. When the cores are set to be shared with windows, they are assigned a limit of how much percentage of the chosen cores are to be available for use. Sharing cores means that windows still has access to the core and will in most cases produce latency to the PLC program as the two systems fight for power. Isolating a core to be used exclusively for TwinCAT means that windows will run a bit slower as it has one fewer core to work with, but in return TwinCAT will have access to 100% of an uninterrupted core. There was an attempt to run TwinCAT with a shared core at 80% usage limit, but after running the system in this configuration for some time and looking at the latency imposed on the system, it was decided to try to run the PLC using dedicated cores instead since it spiked all the time up to about 100μs latency.

3.3 Programming and Implementation in Unreal Engine

3.3.1 Unreal Engine

Unreal Engine is a game engine developed by Epic Games, and comes in five versions, Unreal Engine 1, up to Unreal Engine 5. Unreal Engine 5 recently came out (5. April 2022), and is still a bit too new when it comes to flow of information and guides available. Therefore Unreal Engine 4 which have been out since March 2014, is the chosen version used. Even though this version is a couple of years, it have been updated steadily throughout the years and is regarded by many as one of the best tools for creating great looking high budget games and movies.

Advantages of using Unreal Engine:

- Graphical coding via Blueprint, easy option instead of using C++, or C# if using Unity as engine
- Results in more realistic graphics in general. Getting the same visual fidelity in Unity takes a lot more work
- Completely free to use, where as using the more advanced functions of Unity requires a purchase
- Open-source, while Unity does not allow tinkering with the source code
- Faster rendering times than Unity, which is important in a project such as this where the system will need to be rendered over and over again, while being time efficient

3.3.2 Importing Models

To import models into Unreal Engine, for them to be later placed into the game world, model files for Autodesk were used. The main components needed to visualize the crane had to be converted into a file format that Unreal Engine supported. As Unreal Engine do not support any of the file types from CAD by itself, a plugin named Datasmith had to be used for importing the files into usable mesh files. The part files for Pedestal, King, Main boom, Knuckle jib, hydraulic cylinders and connectors for the outer piston were converted into STEP files, and imported using Datasmith. This created a lot of geometry and texture files, filling the project with too many small parts loaded in, which slowed down Unreal. The solution to this was to highlight all the components that were used to build a part and convert them into one single solid mesh, and then to delete the excess files.

The last step before using these newly imported static meshes, were to assign each of them with collision meshes. The collision mesh is what the physics engine of Unreal uses in combination with the assigned masses, to calculate the center of gravity along with inertia. Without this, the option to activate physics cannot be enabled.

3.3.3 Unreal Setup

After components were imported into Unreal, some had to be attached to each other to create full assemblies. For example the crane which had to be put together piece by piece and then configured further to be able to operate as a crane should. To make the crane into an assembly, all the main components were put together as a Blueprint named BP_{Crane}. Creating a Blueprint for the crane, enables adjustments, assembly and configuration of components within that Blueprint in a closed environment.

Figure 3.9: Picture of the Blueprint for the crane

The hydraulic cylinders used on the crane needed to be put together by the cylinder and the piston, and then had to be applied physics constraints which gave the piston the function of extend and retract within the cylinder housing. The linear motion of the piston then needed to be set to limited and applied a value for how far the piston could move. In Unreal limiting these types of movement constraints means how far the constraint can move in both ways, so the constraint itself needs to be put at mid center of where it should be allowed to move and then set half the stroke length as the value for max movement. This results in the piston being able to move within the correct area and stops the piston from going through the cylinder top or falling out of the cylinder.

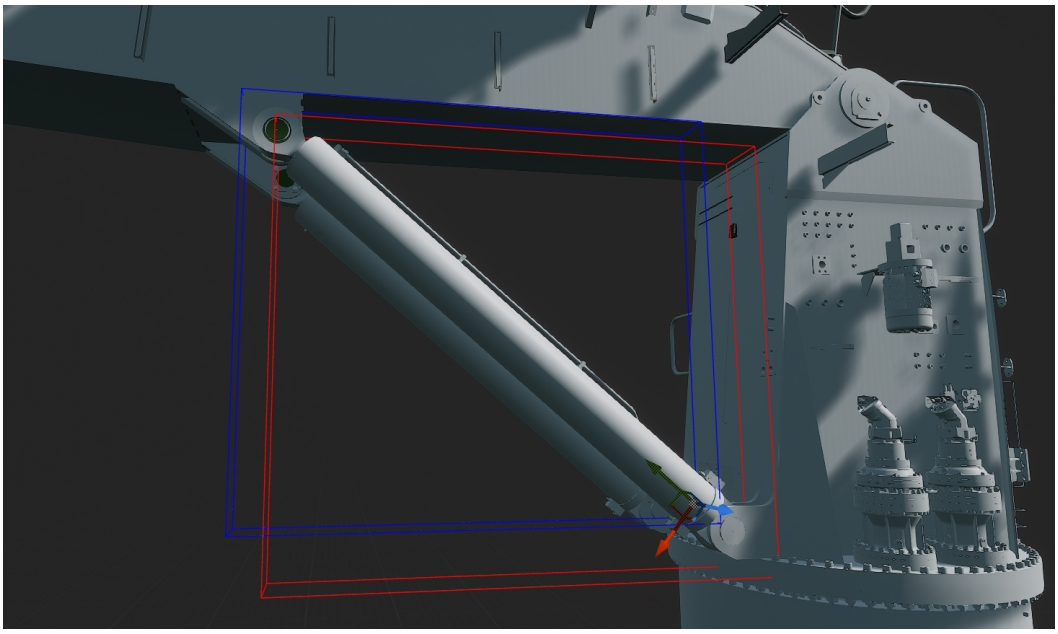


Figure 3.10: Picture of hydraulic cylinder with constraint

After the system components were assembled and put together, the visuals for the simulation were completed. The last thing to do to fully implement the use of Unreal Engine, was to enable communication to the PLC.

Figure 3.11: Picture of the finished Unreal Engine system, showing the boat, ocean, and crane

The assembly of the crane was made using accurate placements of components. For this, dimension drawings were made using the CAD assembly in Inventor.

3.3.4 PhysX - Unreal‘s Physics Engine

For computing collision, behaviour of masses, and forces and reaction forces acting on components, the physics engine PhysX developed by Nvidia is used. This is the default physics engine that Unreal comes with and uses. Currently in the works is a new lightweight physics engine named Chaos, but this is not yet ready for use, as it is still in beta.

3.4 Co-Simulation

3.4.1 TwinCAT Communication

To enable communication between Epic games Unreal Engine and Beckhoffs TwinCAT PLC software, Beckhoffs own free ADS libraries[15] was used. ADS is the communication protocol of TwinCAT and enables among other things access to I/O tasks, Access by variable name, synchronous and asynchronous access, and cyclic and event-based messages. The ADS package was then further developed through C++ programming by Ocean Infinity Marine, and given additional functions which made it able to communicate back and forth with Unreal. The C++ library was then downloaded from their Github site and extracted into the the projects folders where the visual world project was saved, inside a plugin folder. Through the use of Microsoft Visual Studio 2019, an IDE mainly used for software development, the code was compiled and ready to be tested.

For the ADS communication to work correctly and apply values from TwinCAT over to Unreal and vice versa, visual scripting through Unreals Blueprints had to be learned and utilized. The Blueprint code for this takes care of pulling up the ADS master node that receives variable values from TwinCAT and connecting those variables to a set crane component such as the hydraulic pistons positions, velocities, force components etc.

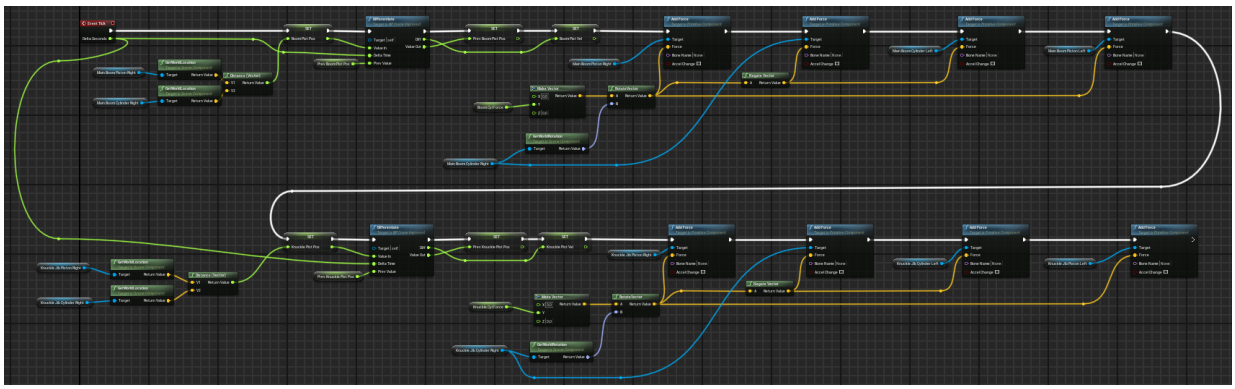


Figure 3.12: Picture of Blueprint code for the hydraulic cylinders

The picture above shows the how the Blueprint boxes are connected, for closer inspection, closeups of the Blueprint code are found in Appendix C C.

To be able to monitor parameter signals sent over the ADS, an interface panel for Unreal were developed and added to the ADS system. Inside Unreal, the values for these can then be set directly via details window found in the editor for the boat and crane system. These parameters point to TwinCAT‘s function blocks as targets and the shows the numerical values for the system as it is running.

Connecting Unreal Engine to TwinCAT means that the PLC does not need to calculate position or velocity anymore, since these can now come from Unreal Engine‘s built in "get position" and "get velocity" functions through the use of Blueprints. So now the system works as follows, piston forces for the hydraulic cylinders are sent over ADS to Unreal Engine, to move the arms of the crane. The movements of the arms are then sent back from Unreal over to TwinCAT where calculations regarding required force are continuously being calculated depending on the movement.

3.5 Testing

3.5.1 MATLAB/Simulink Simulation

For cross system comparison to the PLC, and to be able to verify if the programmed PLC system was working as intended, a Simulink multibody model was created. The multibody system uses rigid transforms to place revolute and sliding joints in their respective coordinate and rotational positions, using measurements from CAD assembly of the crane. The CAD files of the crane bodies uses CAD models supplied by Ocean Infinity Marine, were then converted into stl file format which describes the surface geometry of a model. These were then used to visualize the crane using solid blocks in Simulink and replacing the solid block models with the stl models. The crane could then be seen within MATLAB's mechanics explorer window and confirmed to be put together correctly.

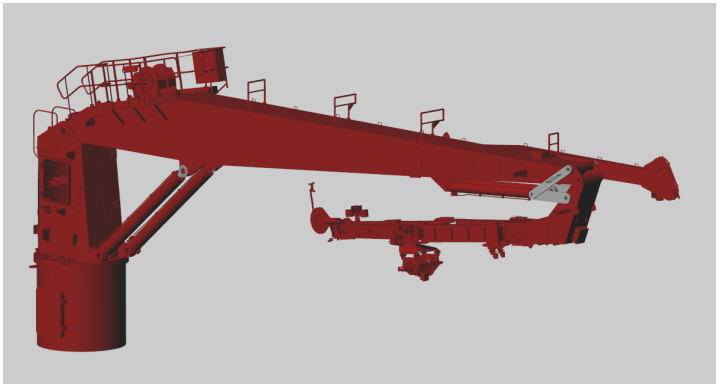


Figure 3.13: The modeled crane in Simulink, shown in the mechanics explorer of MATLAB

A hydraulic control system was then implemented into the Simulink model, using the same mathematical setup and programming that is in the PLC, to create an identical system. The control system was made using MATLAB functions within Simulink, and split into three subsystems. One subsystem for controlling the slew joint using the hydraulic motor, and two subsystems for controlling the sliding joints at the main boom and knuckle jib, using hydraulic actuators. Parameters used for this system are copied from the parameter STRUCT found in the PLC.

The Simulink joints were then configured to be able to take torque and force from the control system subsystems as inputs and then automatically compute movement changes accordingly. The sense feature of the joints were activated for both position and velocity, which values were sent back to the hydraulic control system and used to continuously calculate forces, pressures, flow, etc to ensure smooth movement of the crane.

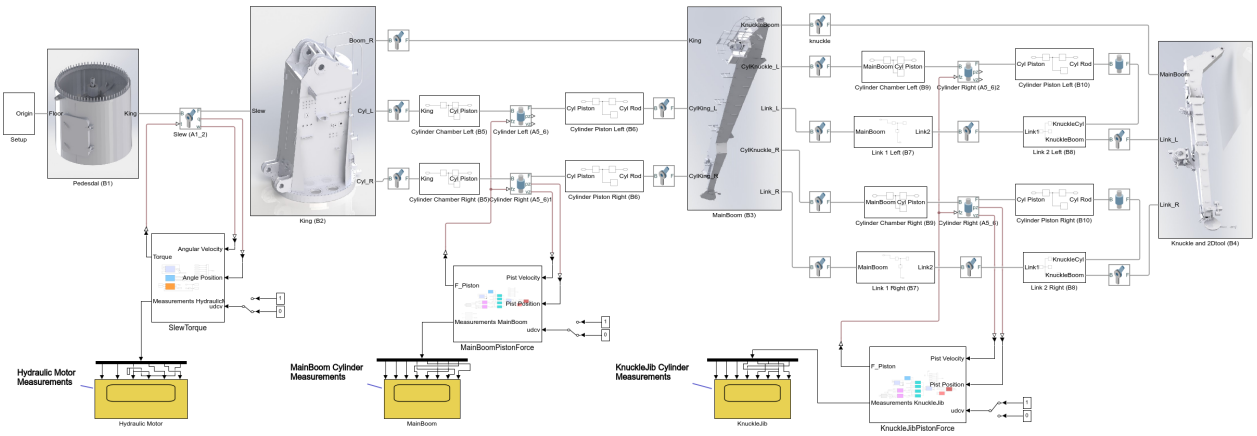


Figure 3.14: Overview of the Simulink model

The completed Simulink system can be seen above. This is the uppermost layer of the model, while the contents of each subsystems and their setup can be found in the Appendix DD.

3.5.2 TwinCAT Measurements

To be able to monitor and get good clean overviews of the various variables as the system is running, TwinCAT's own measurement project program was used. This is run as a separate instance to the PLC software, but is still integrated closely to the PLC running. Through creating a measurement project, the variables in the PLC that are desirable to be monitored can be found and pointed to via hierarchy view within TwinCAT measurement. The variables can then be recorded as the system runs, and can be used to compare improvements to the system as project progresses.

3.5.3 Test #1 - Simulink vs TwinCAT

To deduct if the hydraulic system was working as intended, an hydraulic simulation and control subsystems was extracted from the Simulink multibody simulation system. This was then placed in its own Simulink model to be used without a connection to the multibody model. Instead this was connected to a Matlab function block containing equation 2.1 set up in section 2.2.1. The same was done in TwinCAT, where the system ran independently with no connection to the multibody model made in Unreal Engine. This was done to eliminate possible sources of error, to be able to check the pure hydraulic calculations.

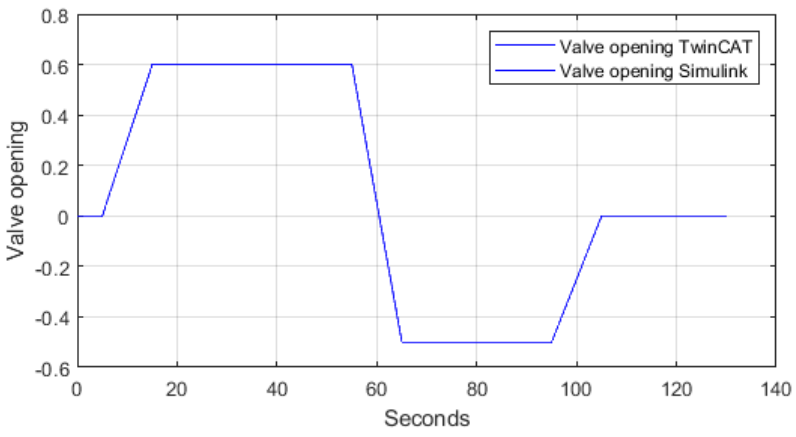


Figure 3.15: The ramp ups, hold, and ramp downs used for initial testing

To give the systems directions for running the test, a ramp up and down function was set up in both Simulink and in TwinCAT PLC. This was then used as the opening value for the valve, with the goal of making the hydraulic pistons extrude and retract. Figure 3.15 shows the ramp up/down targets.

3.5.4 Test #2 - Simulink vs TwinCAT & Undreal Engine

For this test setup, the Simulink multibody model was connected to the hydraulics simulation and control system, as seen in chapter 3.5.1, figure 3.14. The test to be run was fairly simple, but included use of the limit forces made in equations 2.9 for and 2.10 for simulating the piston hitting the top or bottom inside the hydraulic housing. The test was to run using a completely open valve, to reach the maximum position of each arm tested separately, and to be able to stop there via forces negating further movement.

Chapter 4

Results

This chapter presents the outcome of the research and development done during the thesis. The results of running the crane to test the hydraulic system and the dynamic movement of the arms, were done using both the multibody system in Simulink, and the multibody system in Unreal Engine combined with TwinCAT. While actuating the crane, resulting variables for flow, pressure, velocity, positions and force were measured. The measurements taken were then used to compare the behaviour of the two crane systems.

The two multibody systems has some similarities, whereas they both use the same equations and parameters to complete the hydraulic simulation aspect. Their major difference that makes it interesting for comparison however, is their different implementation of the hydraulic simulation and control system.

Simulink and its multibody system actuates via an hydraulic system placed in the same Simulink environment as the multibody model. Making it into a complete package where all computations of both hydraulics and dynamics happen without delay in one and same software.

TwinCAT on the other hand, incorporates the use of a second software, the Unreal Engine, to simulate the crane system. The simulation of an hydraulic system happens in TwinCAT, while the dynamics are being computed within Unreal Engine, where the crane is also visualized.

The TwinCAT+Unreal Engine system works by sending a piston force from TwinCAT to Unreal over ADS to actuate each hydraulic cylinder. In return, movement data are sent back from Unreal Engine into TwinCAT. This back and forth handling of variables, is how the HIL simulation works.

4.1 Test #1

Comparisons of test scenario 1 where there were no multibody systems connected to TwinCAT or Simulink, are shown in this section, this test was run using only the main boom as it was to test if the hydraulic system was working as intended. The valve opening was controlled with a ramp up/down input as seen in section 3.5.3 figure 3.15.

4.1.1 Hydraulics

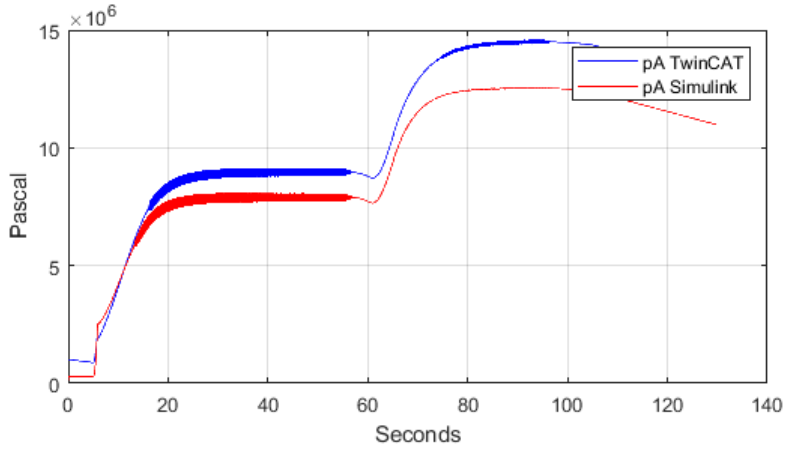


Figure 4.1: Comparison of pressure pA

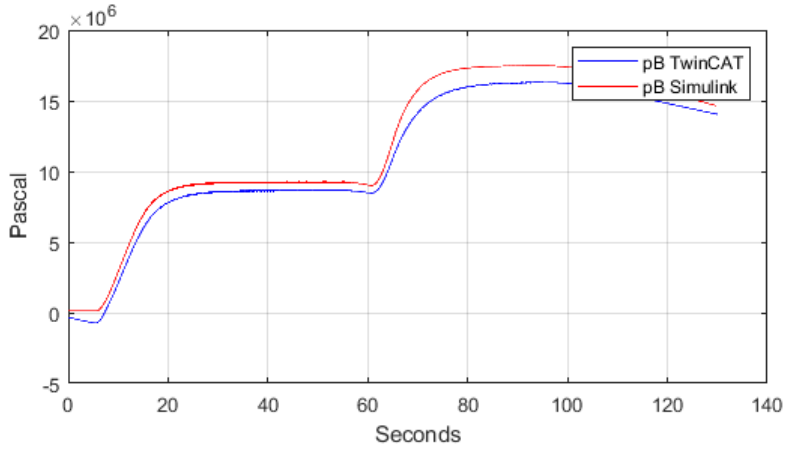


Figure 4.2: Comparison of pressure pA

As noticed by inspecting the results from figure 4.1 and 4.2, there is some difference, although not a large one. This is suspected to be because of the initial value for pressure. It is set as a start value for both systems. In Simulink it is set as an initial value inside an integrator simscape block, while in TwinCAT it is set as an initial value in the programmed integrator function block. There may be a difference how the blocks interprets this.

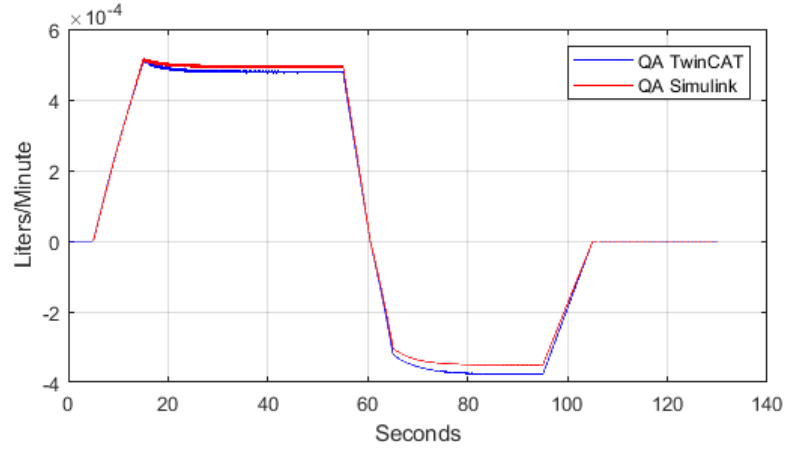


Figure 4.3: Comparison of flow QA

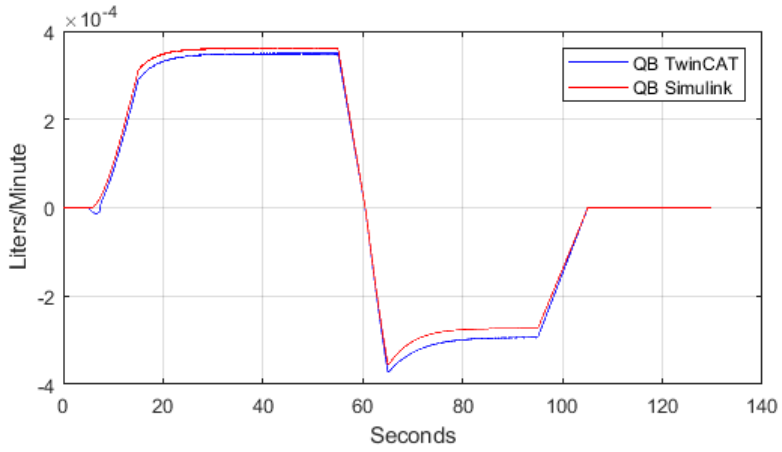


Figure 4.4: Comparison of flow QA

The hydraulic flows shown in figures 4.3 and 4.4 closely resembles each other, and in the ramp up and down sections they are almost inseparable.

4.1.2 Mechanical

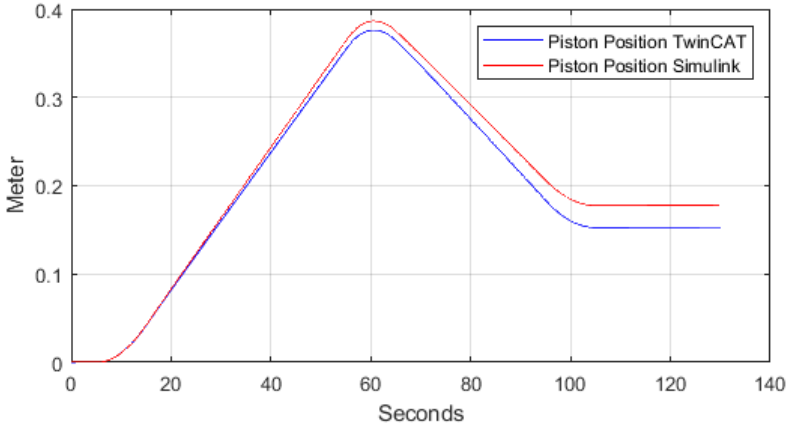


Figure 4.5: Comparison of position

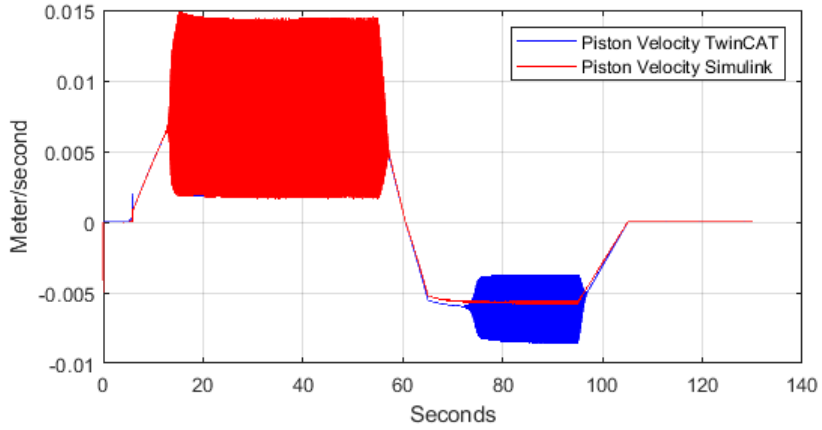


Figure 4.6: Comparison of velocity

It can be seen from the motions of the piston in figure 4.5 and 4.6, that the movement of the pistons in the two systems performs similar to each other, although with some deviations.

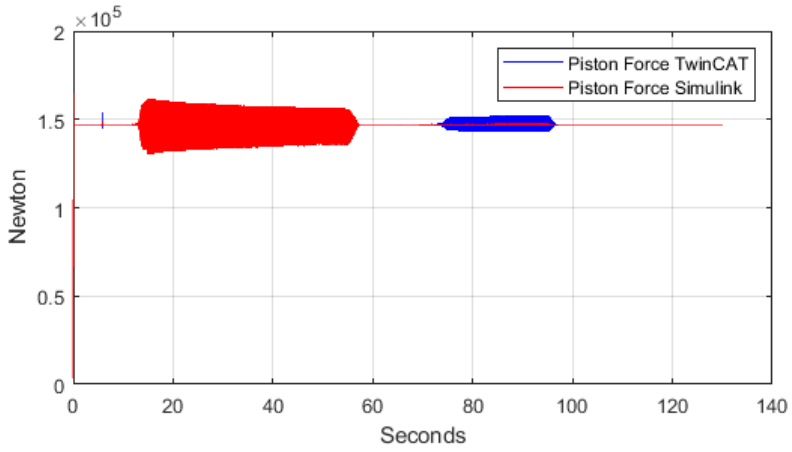


Figure 4.7: Comparison of piston force

The piston force shown in figure 4.7, shows behaviour that matches with the outputs shown in figure 4.6 for velocity.

4.2 Test #2

As described in 3.5.4, this test uses the complete system combining both TwinCAT Unreal Engine to run the simulation. For comparison the Simulink multibody system was also used with the same setup of hydraulic actuation.

4.2.1 Main Boom

Hydraulics

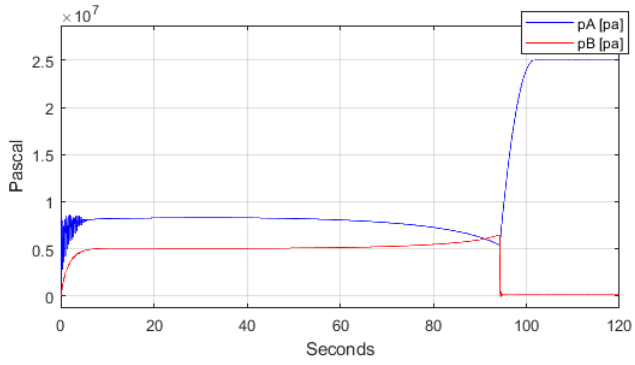


Figure 4.8: Pressure in (pA) and out (pB) from running the main boom using the Simulink model

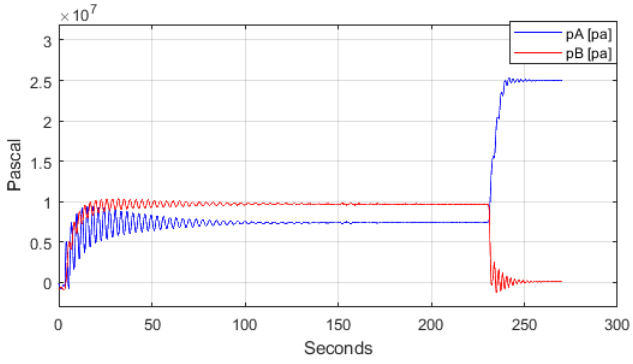


Figure 4.9: Pressure in (pA) and out (pB) from running the main boom using the real-time TwinCAT system

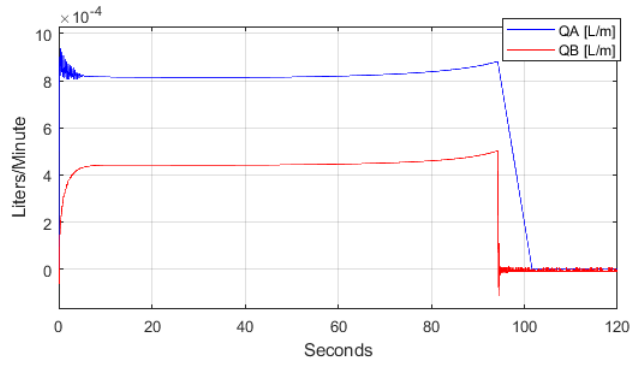


Figure 4.10: Flowrates QA and QB from running the main boom using the Simulink model

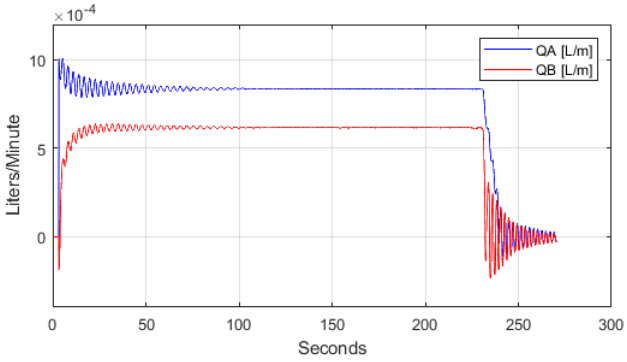


Figure 4.11: Flowrates QA and QA from running the main boom using the real-time TwinCAT system

Mechanical

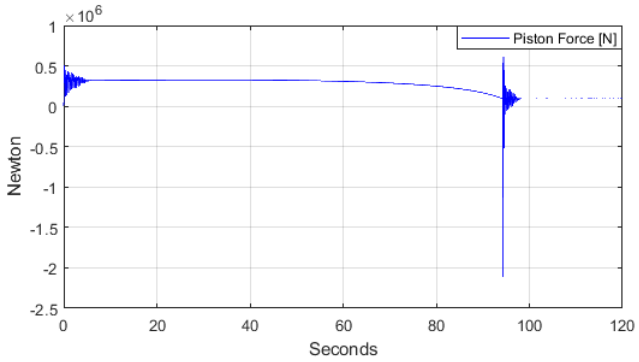


Figure 4.12: Piston force acting on the main boom using the Simulink model

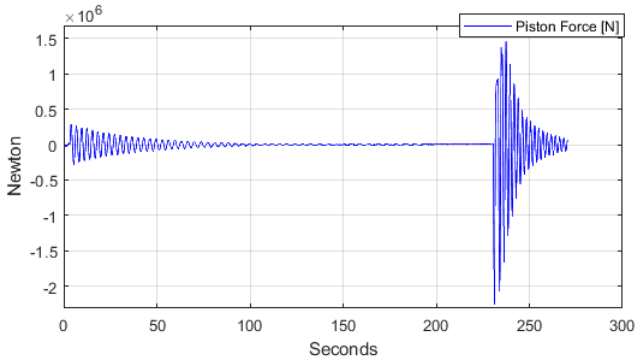


Figure 4.13: Piston force acting on the main boom using the real-time TwinCAT system

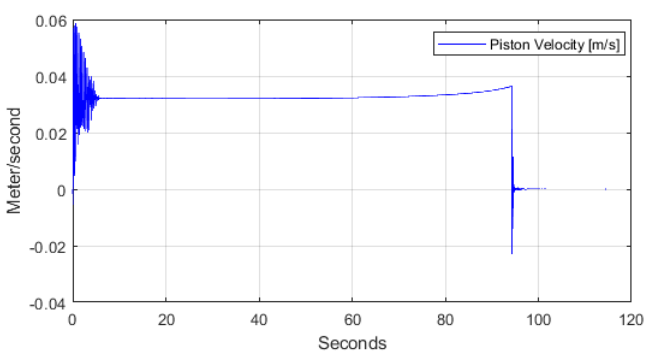


Figure 4.14: Velocity of the piston when running the main boom using the Simulink model

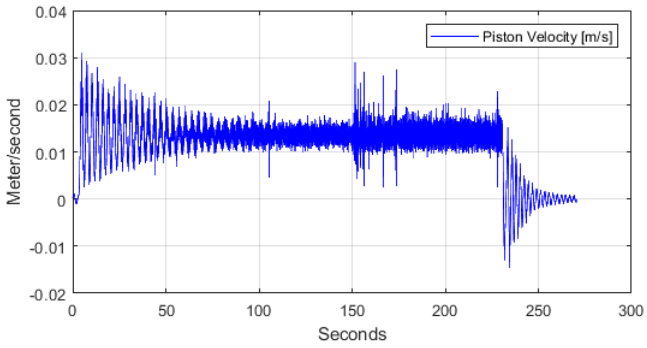


Figure 4.15: Velocity of the piston when running the main boom using the real-time TwinCAT system

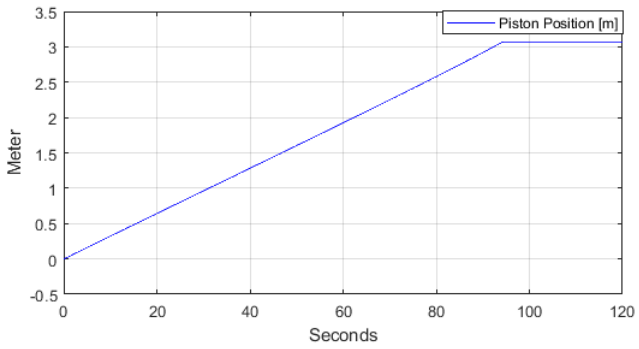


Figure 4.16: Position of the piston pushing the main boom using the Simulink model

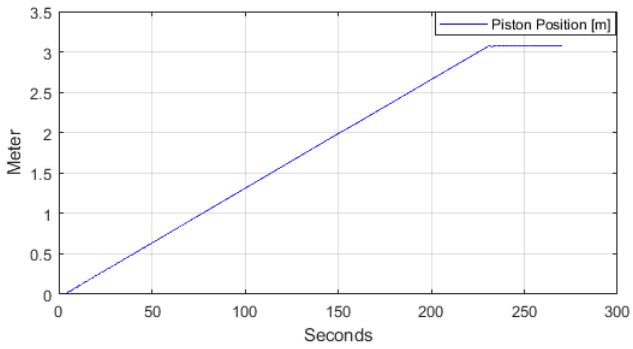


Figure 4.17: Position of the piston pushing the main boom using the real-time TwinCAT system

Looking at the plotted results from both the Simulink model and the real-time PLC with Unreal engine using its PhysX engine for computing physics, it can be seen that there are big differences between the two systems even though the system that controls the crane are basically the same.

This is most likely a result of the way physics are handled in Unreal Engine. The physics is tied to the framerate that the PC running is capable of delivering [5].

Both systems managed to reach their desired cylinder stroke length, moving the crane's main boom up to its maximum position at 3.07 [m], and then stabilizing at that point without having to put on some kind of saturation limit to the system. To reach the position, the Simulink system used approximately 92 seconds, while the PLC system used approximately 230 seconds, using 2.5 times more time than Simulink.

4.2.2 Knuckle Jib

Hydraulics

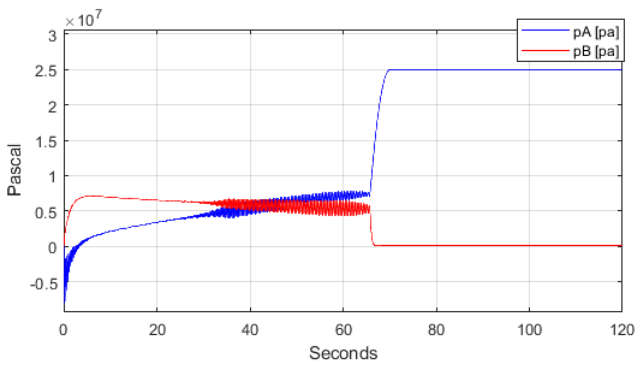


Figure 4.18: Pressure in (pA) and out (pB) from running the knuckle jib using the Simulink model

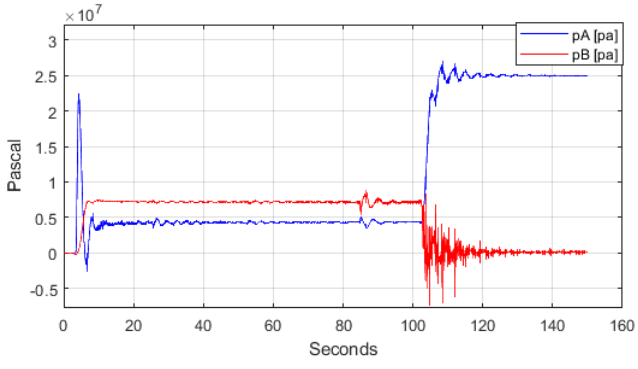


Figure 4.19: Pressure in (pA) and out (pB) from running the knuckle jib using the real-time TwinCAT system

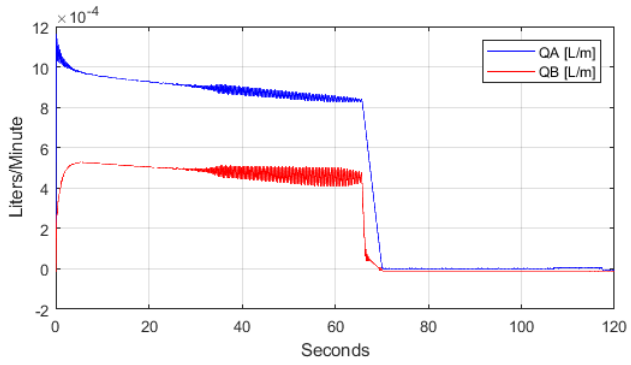


Figure 4.20: Flowrates QA and QB from running the Knuckle jib using the Simulink model

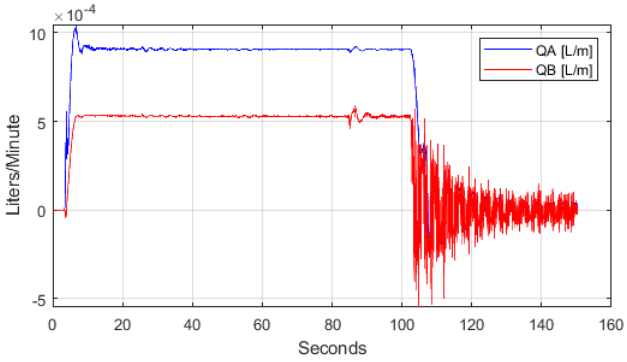


Figure 4.21: Flowrates QA and QA from running the knuckle jib using the real-time TwinCAT system

Mechanical

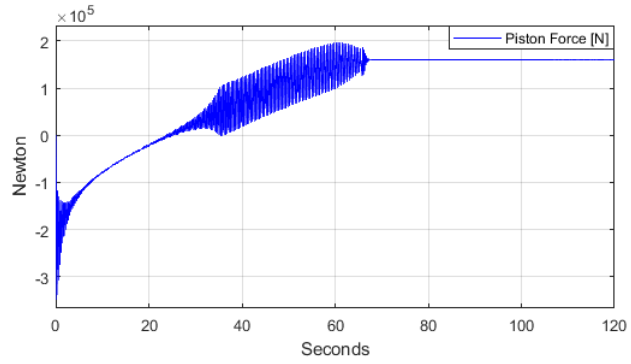


Figure 4.22: Piston force acting on the knuckle jib using the Simulink model

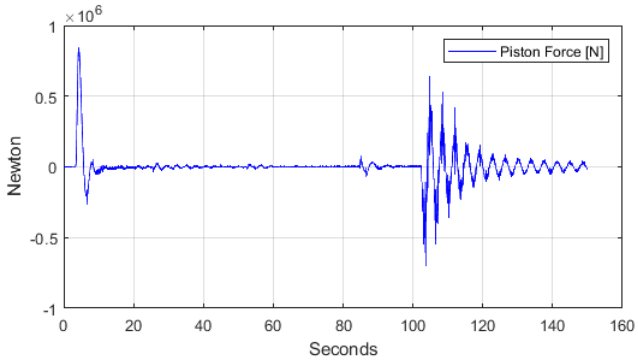


Figure 4.23: Piston force acting on the knuckle jib using the real-time TwinCAT system

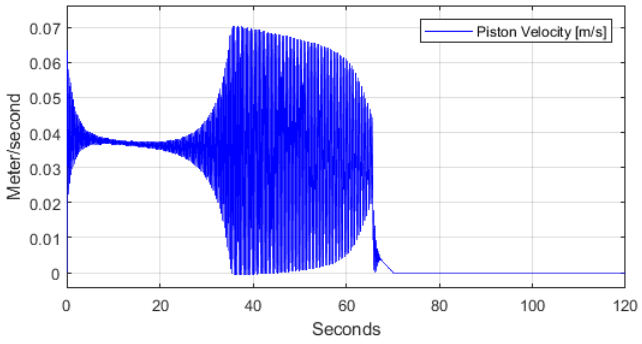


Figure 4.24: Velocity of the piston when running the knuckle jib using the Simulink model

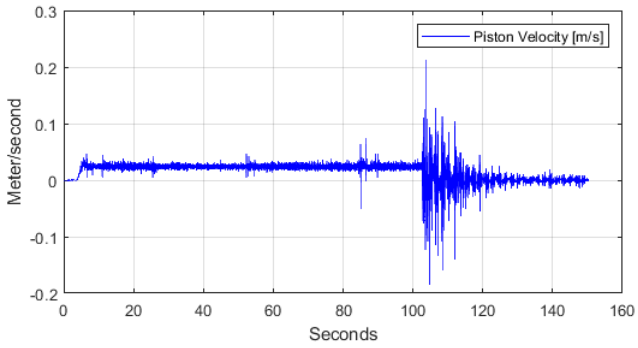


Figure 4.25: Velocity of the piston when running the knuckle jib using the real-time TwinCAT system

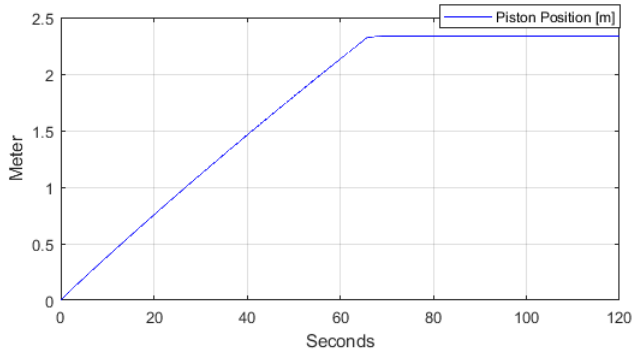


Figure 4.26: Position of the piston pushing the knuckle jib using the Simulink model

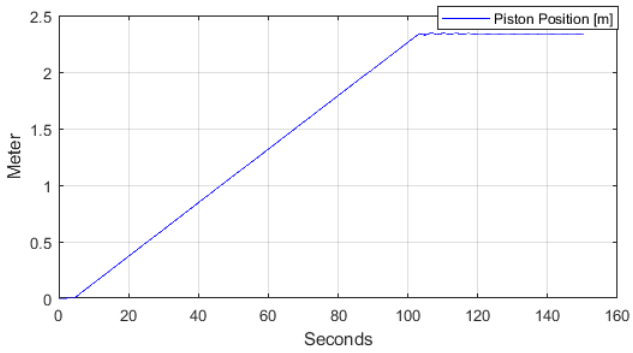


Figure 4.27: Position of the piston pushing the knuckle jib using the real-time TwinCAT system

The differences in timing for the knuckle jib to reach its intended position of max reach, was smaller compared to running the main boom. Using Simulink the knuckle jib used approximately 65 seconds while the knuckle jib running via the PLC used approximately 103 seconds, making it 1.58 times slower than its Simulink counterpart.

Both the Simulink system and the PLC+Unreal system managed to reach their desired stroke length of max position at 2.32m, moving the crane‘s knuckle jib arm up to its maximum position, and then stabilizing at that point without having to put on some kind of saturation limit to the system. To reach the position, the Simulink system used approximately 92 seconds, while the PLC system used approximately 230 seconds, using 2.5 times more time than Simulink.

4.3 Latency

Assigning one core as exclusive improved latency a lot and was deemed well within tolerable levels. The latency stayed mostly below 8μs, although with some deviations at certain stress points. Overall it yielded a much better total latency level.

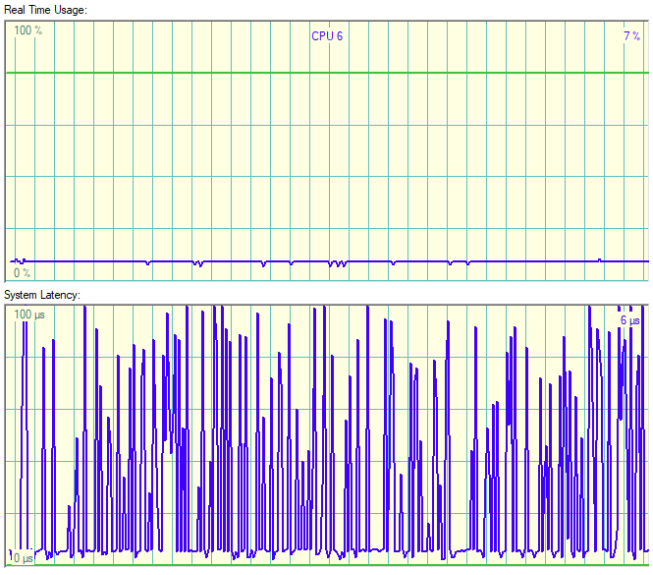


Figure 4.28: Latency and core usage of TwinCAT using a shared core at 80% usage limit

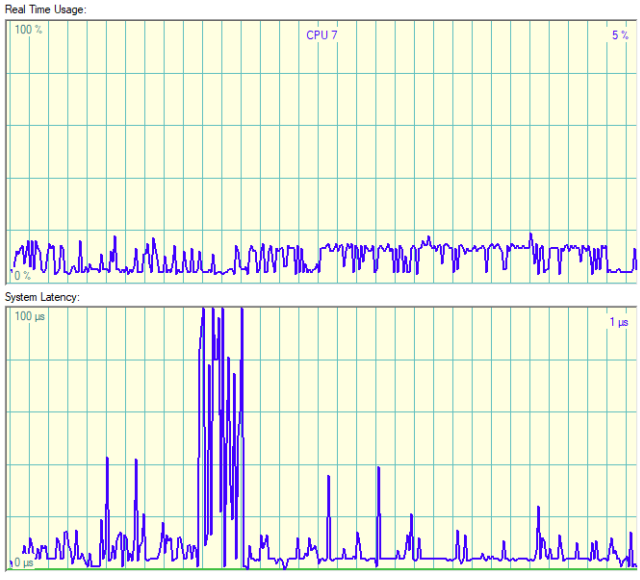


Figure 4.29: Latency and core usage of TwinCAT using a dedicated core

Chapter 5

Discussions

5.1 Differences in Results

As the TwinCAT+Unreal system and the Simulink multibody system produced a large difference in results, where one system used substantially more time than the other, some thoughts about the reasons for this was made. These include the following.

- The difference in physics computing, with Unreal's physics is mainly focused on use within games and perhaps not completely realistic scenarios, and MATLAB/Simulink which is targeted at engineering and research.
- Delay between signals sent over the ads could introduce instability to the system, which is also perhaps why the plots from the PLC system are more sporadic, whereas the plots from Simulink are consistent and vibrations are registered at quicker smoother intervals.
- Density calculated by Unreal could be wrong, as this is calculated automatically based on the set mass and the simple collision geometry of each component which could be inaccurate, without any means of setting them manually.
- Unreal Engine's physics are framerate based, and the computational power of the computer running the game engine, has a direct impact

Unreal Engine

Unreal Engine 4 runs using a variable frame rate which the physics computation is tied to. Meaning that feeding forces to Unreal Engine has very different outcomes depending og the computational power of the hardware it is run on. An example of this can be sending set torque to Unreal Engine running on a high-end computer reaching 120 frames per second, could end up making an object turn around multiple times. While making the same object rotate slightly on a low-end computer only producing a fraction of the same frame rate [9].

5.2 Future Work

There are improvements that can be made to further make the system into a more complete digital twin of the C1079 crane, these can be both minor and major changes and additions, and will be brought up in the subsections below.

Physics simulations in the PLC

A multibody system that was to be provided by Ocean Infinity, was unfortunately not developed in time for the project to utilize it. However this system would be a part of the PLC program, and would be able to calculate the placements and rotations of each

component of the crane. This would eliminate the delay between software as all calculations are done within the PLC and then the Unreal engine part would only be for visualizing, not for calculating physics and computing movement. This would also eliminate the time used for placing crane components and connecting each part together manually in the editor. This will be added in the future as the project is still to be taken to its final form via Ocean Infinity‘s development team.

Inverse kinematics

Using inverse kinematics would be preferable, as it would ease the control aspect of the system. Adding this would let the crane be controlled by tool point coordinates and then adjusts each joint and actuators to reach the desired point.

Waves and wave compensation

To better simulate the crane in the scenario of being placed on a boat, there needs to be waves affecting the movement and rotations of the crane. These waves would need to introduce movement to the boat in six degrees of freedom for it to mimic the real behaviour of a boat on the water. These motions would then need to be implemented into the kinematics to be used as motion compensation of the tool point of the crane.

Some development was done within this field and the boat is able to float on water using a combination Unreal‘s water physics plugin for and buoyancy points added to various locations around the hull of the boat. These settings will need to be tweaked in the future for it to become more realistic in its behaviour.

Chapter 6

Conclusions

The research question proposed was to evaluate if TwinCAT could be used to simulate the hydraulic actuation of a knuckle boom crane in combination with a multibody system in Unreal Engine in real-time. The concrete objectives for acquiring this was: 1) Build a simulator of actuation systems in TwinCAT using the Structured Text PLC programming language; 2) Build a multibody simulation able to compute dynamics in Unreal Engine; 3) Make TwinCAT and Unreal Engine able to communicate; 4) Performance testing with comparison against a Simulink simulation.

A simulation system was built using Beckhoff's TwinCAT PLC, written using object-oriented programming in Structured Text. Further on, the TwinCAT simulation was connected to a multibody system in Unreal Engine via an ADS plugin for sending variables for piston force, positions, and velocity between the two software.
Testing the performance of the electro-hydraulic actuation system using systems separate from the multibody models yielded promising results. However, running the system in co-simulation with Unreal Engine introduced large deviations, which are strongly believed to directly result from the physics being tied to the variable frame rate.

Bibliography

[1] M. Ambrosino and E. Garone. "Modelling and control of a knuckle boom crane." In: *International Journal of Control* 0.0 (2021), pp. 1–18. DOI: 10.1080/00207179.2021.1913290. URL: https://doi.org/10.1080/00207179.2021.1913290.

[2] Morten K. Bak. "Model based design of electro-hydraulic motion control systems for offshore pipe handling equipment." In: 2014.

[3] Ben. *ads-xbox-controller*. https://github.com/benhar-dev/ads-xbox-controller. 2021.

[4] Darren Cottingham. *Telescopic boom or knuckle boom crane*. URL: https://www.drivingtests.co.nz/resources/telescopic-boom-crane-vs-knuckle-boom-crane/. (accessed: 07.06.2022).

[5] Epic Games. *Physics Sub-Stepping*. URL: https://docs.unrealengine.com/4.27/en-US/InteractiveExperiences/Physics/Substepping/. (accessed: 10.06.2022).

[6] Daniel Hagen. "Improving Energy Efficiency and Motion Control in Load-Carrying Applications using Self-Contained Cylinders." PhD thesis. University of Agder, 2020. URL: https://uia.brage.unit.no/uia-xmlui/handle/11250/2673158.

[7] *IEC 61131-3:2013 Programmable controllers - Part 3: programming languages*. en/fr. International Standard ISO/IEC TR 29110-1:2016. International Electrotechnical Commission, 2013. URL: https://webstore.iec.ch/publication/4552.

[8] Beckhoff Automation GmbH Co. KG. *TwinCAT 3: the flexible software solution for PC-based control*. URL: https://www.beckhoff.com/media/downloads/informationsmedien/beckhoff_twincat3_e.pdf. (accessed: 10.06.2022).

[9] Víctor Ávila Parcet. *Framerate independent physics in UE4*. URL: https://avilapa.github.io/post/framerate-independent-physics-in-ue4/#understanding-the-problem. (accessed: 10.06.2022).

[10] R Rahmfeld. "Development and control of energy saving hydraulic servo drives for mobile systems." PhD thesis. July 2002.

[11] Red Rock. *Telescopic boom or knuckle boom crane*. URL: https://www.redrock.no/portfolio/knuckle-boom-crane-rck-orck/. (accessed: 08.06.2022).

[12] Red Rock. *Telescopic boom or knuckle boom crane*. URL: https://www.redrock.no/portfolio/telescopic-boom-crane-rct-orct/. (accessed: 08.06.2022).

[13] Jakob Sagatowski. *The five best and worst things with TwinCAT*. URL: https://alltwincat.com/2020/06/15/the-five-best-and-worst-things-with-twincat/. (accessed: 09.06.2022).

[14] Harald Sangvik. "Digital Twin of 3d Motion Compensated Gangway Use of Unity Game Engine and TwinCAT PLC Control for Hardware-in-the-Loop Simulation." In: 2021.

[15] Beckhoff New Automation Technology. *TC1000 TwinCAT 3 ADS*. URL: https://www.beckhoff.com/nn-no/products/automation/twincat/tc1xxx-twincat-3-base/tc1000.html. (accessed: 14.05.09.2022).

Appendix A

Datasheet A

Task given for the master thesis

UiA Faculty of Engineering and Science

RED ROCK

Draft: Development of HIL Simulator for Autonomous Load Handling Equipment

"Use of Unity game engine and TwinCAT real-time control system for hardware-in-the-loop testing and virtualization"

Company	Project Level	NDA?
Red Rock Marine AS	Master	No

Background

Red Rock Marine is the leading supplier of the next generation of digital lifting and handling systems for the offshore and marine markets. We are currently focusing our research on autonomous load handling systems. Therefore, we are continuously developing our simulation environment (Digitial Twins) to validate our control system and test new control algoritms.

Project Description

Red Rock has developed a Digital Twin of a 3D motion compensated gangway running real-time in Unity. This project aims to improve the existing virtual simulator and integrate realistic Mechatronic simulation models of our 3D compensated knuckle boom crane in our Beckhoff TwinCAT real-time control system to simplify implementation and improve software testing.

Click here to see the movie of our previous MSc project

Keywords

- PLC Programming (Structured Text)
- User Interface (HMI)
- Modeling and Simulation of Mechatronic Systems
- Model-based Design
- 3D Motion Compensation (wave spectrum simulation)
- Hardware-in-the-loop (HIL) Simulation/Testing
- Real-time Control Systems

Click here to se movie of our famous 3D crane

Additional Information

Red Rock has a simulator dome available where the simulation environment can be demonstrated. Beckhoff TwinCAT PLC and HIL-PC can be provided for running the control system and the HIL Simulator.

Contact information:

Full name	E-mail address	Phone number
Harald Sangvik, MSc (Control System Engineer)	harald.sangvik@redrock.no	93820535
Daniel Hagen, PhD (Senior R&D Engineer)	daniel.hagen@redrock.no	92013462

Appendix B

Datasheet B

TwinCAT code

Listing B.1: Main (PRG)

```
Timer := Timer + 0.001;

ActuationSystems();
P_Joy();
```

Listing B.2: ActuationSystems (PRG)

```
fbHPU(
);


fbramp(
    bOn := startramp,
    y0 := 0,
    y1 := 0.6,
    y3 := -0.5,
    t0 := 5,
    t1 := 15,
    t2 := 55,
    t3 := 65,
    t4 := 95,
    t5 := 105,
    t6 := 120,
);

fbFWKinematics(
    L1 := 4.543126+1.097+1.000,                                          ...
                //[m]
    L2 := 0,                                                             ...
                //[m]
    L3 := 17.939373,                                                     ...
                //[m]
    L4 := 9.548784,                                                      ...
                //[m]
    thetaMainBoom := ...
       ActuationSystems.fbKnuckleJibCylinderHyd.AngleJoint,       //[rad]
    thetaKnuckleBoom := ActuationSystems.fbMainJibCylinderHyd.AngleJoint, ...
             //[rad]
    thetaSlew := 0                                                       ...
                //[rad]
);

//Hyd Model for MainJibCylinder
```

```
fbMainJibCylinderHyd(
    uDcvEnable := Inputs.uDcvEnableMainJib,
    uDcv := Inputs.uDcvMainJib,
    ReturnPressure := G_ActuationSystems.ReturnPressure,
    SupplyPressure := G_ActuationSystems.SupplyPressure,
    par := Param.pMainJibCylinder,
    xPiston := work.XpistonMainBoom,
    vPiston := work.VpistonMainBoom,
);
//Hyd Model for KnuckleJibCylinder
fbKnuckleJibCylinderHyd(
    uDcvEnable := Inputs.uDcvEnableKnuckleJib,
    uDcv := Inputs.uDcvKnuckleJib,
    ReturnPressure := G_ActuationSystems.ReturnPressure,
    SupplyPressure := G_ActuationSystems.SupplyPressure,
    par := Param.pKnuckleJibCylinder,
    xPiston := work.XpistonKnuckleJib,
    vPiston := work.VpistonKnuckleJib,
);
//Hyd Model for KingRotationCylinder
fbKingRotationCylinderHyd(
    uDcvEnable := Inputs.uDcvEnableKingRotation,
    uDcv := Inputs.uDcvKingRotation,
    ReturnPressure := G_ActuationSystems.ReturnPressure,
    SupplyPressure := G_ActuationSystems.SupplyPressure,
    par := Param.pMainJibCylinder,
    xPiston := fbkingRotationCylinderMech.PistonPosition,
    vPiston := fbkingRotationCylinderMech.PistonVelocity,
);


//Temp Mechanical Model for MainJibCylinder
fbMainJibCylinderMech(
    F_Hydraulic := fbMainJibCylinderHyd.F_Hydraulic,
    param := Param.pMainJibCylinder,
    F_Piston => Work.FpistonMainBoom,
    PistonVelocity := work.VpistonMainBoom,
    PistonPosition := work.XpistonMainBoom,
    Spring := 6E9,
    Damper := 2E7
);
//Temp Mechanical Model for KnuckleJibCylinder
fbKnuckleJibCylinderMech(
    F_Hydraulic := fbKnuckleJibCylinderHyd.F_Hydraulic,
    param := Param.pKnuckleJibCylinder,
    F_Piston => Work.FpistonKnuckleJib,
    PistonVelocity := work.VpistonKnuckleJib,
    PistonPosition := work.XpistonKnuckleJib,
    Spring := 5.1E7,
    Damper := 0
);
//Temp Mechanical Model for KingRotationCylinder
fbkingRotationCylinderMech(
    F_Hydraulic := fbKingRotationCylinderHyd.F_Hydraulic,
    param := Param.pKingRotationCylinder,
    F_Piston => Work.FpistonKingRotation
);
```

Listing B.3: ValveControlledCylinder (FB)

```
//Cylinder Areas [m^2]
```

```
PistonArea := EXPT(par.DBore,2) * (par.pi/4.0);
AnnulusArea := (EXPT(par.DBore,2) - EXPT(par.DRod,2)) * (par.pi/4.0);

//ValveOpeningControl := XboxControl.joyLeft;      Commented out xbox ...
   control as for now

//ValveDynamics [-1,1]
IF uDcvEnable THEN
    fbValveDynamics(
        udcv_IN := uDcv,
        DynamicValve => ValveOpeningControl
        );
ELSE
    fbValveDynamics(
        udcv_IN := 0,
        DynamicValve => ValveOpeningControl
        );
END_IF

//VariableBulkModulus [Bar]
fbBulkModulus(
    OilBulkModulus := par.OilBulkModulus,
    AdiatricAirConstant := par.AdiatricAirConstant,
    AtmosphericPressure := par.AtmosphericPressure,
    VolumetricAirContentOfOil := par.VolumetricAirConstant
    );


//Pressure Gradient pA and Pressure pA [Pa]
fbPressureGradient_pA(
    //TotalVolume := par.VA0 + (xPiston * PistonArea),
    TotalVolume := 0.01 + (xPiston * PistonArea),
    BulkModulus := fbBulkModulus.BulkModulus,
    FlowrateIn := FlowQA,
    FlowrateOut := (vPiston * PistonArea),
    //p0 := par.pA0,
    p0 := 0,
    param := par,
    Pressure => PressurepA
    );

//Pressure Gradient pB and Pressure pB [Pa]
fbPressureGradient_pB(
    //TotalVolume := par.VB0 + ((par.xMax - xPiston) * AnnulusArea),
    TotalVolume := 0.01 + ((par.xMax - xPiston) * AnnulusArea),
    BulkModulus := fbBulkModulus.BulkModulus,
    FlowrateIn := (vPiston * AnnulusArea),
    FlowrateOut := FlowQB + FlowQL,
    //p0 := par.pB0,
    p0 := 0,
    param := par,
    Pressure => PressurepB
    );

//Flow through orifices Q_P_A, Q_A_R, Q_B_R, Q_P_B [m^3/s]
fbOrificeEquationQ_P_A(
    PressureOut := PressurepA,
    PressureIn := SupplyPressure,
    ValveOpening := ValveOpeningControl,
    param := par,
    OrificeFlow => FlowQ_P_A
```

```
    );

fbOrificeEquationQ_A_R(
    PressureOut := ReturnPressure,
    PressureIn :=  PressurepA,
    ValveOpening := ValveOpeningControl,
    param := par,
    OrificeFlow => FlowQ_A_R
    );

fbOrificeEquationQ_B_R(
    PressureOut := ReturnPressure,
    PressureIn := PressurepB,
    ValveOpening := ValveOpeningControl,
    param := par,
    OrificeFlow => FlowQ_B_R
    );

fbOrificeEquationQ_P_B(
    PressureOut := PressurepB,
    PressureIn := SupplyPressure,
    ValveOpening := ValveOpeningControl,
    param := par,
    OrificeFlow => FlowQ_P_B
    );

//Change direction of flow depending on valveopening
IF ValveOpeningControl > 0 THEN
    FlowQA := FlowQ_P_A;
    FlowQB := FlowQ_B_R;
ELSIF ValveOpeningControl < 0 THEN
    FlowQA := FlowQ_A_R;
    FlowQB := FlowQ_P_B;
ELSE
    FlowQA := 0.0;
    FlowQB := 0.0;
END_IF

// Hydraulic Force [N] //
F_Hydraulic := PressurepA * PistonArea - PressurepB * AnnulusArea;

//Crane angles
fbCraneAngleJoint(
    a := (par.LengthCylinder + xPiston),
    b := (par.LengthAdjacentLeft),
    c := (par.LengthAdjacentRight),
    thetaAngle => AngleJoint
);

//Hydraulic Motor
fbHydraulicMotor(
    PressureIn := PressurepA,
    PressureOut := PressurepB,
);
```

Listing B.4: PressureGradient (FB)

```
//PressureGradient in [Pascal/s]
PressureGradient := (FlowrateIn - FlowrateOut) * (BulkModulus / TotalVolume);

//Pressure in [Pascal]
```

```
fbPressurePascal(
    ssMethodType := ssMethodType,
    tCycle := param.tCycle,
    In := PressureGradient,
    Init := p0,
    //minOut := G_ActuationSystems.ReturnPressure,
    minOut := 0,
    //maxOut := param.pAmax,
    maxOut := 10000000000000000,
    Out => Pressure
    );
```

Listing B.5: CylinderMech (FB)

```
// Upper Stroke Limit Contact Force [N] //
IF PistonPosition >= param.xMax THEN
    SpringContactCoefficient := Spring;
ELSIF PistonPosition < param.xMax THEN
    SpringContactCoefficient := 0;
END_IF;

IF PistonPosition >= param.xMax THEN
    DamperContactCoefficient := Damper;
ELSIF PistonPosition < param.xMax THEN
    DamperContactCoefficient := 0;
END_IF;

F_Upper_Stroke_Limit := ((PistonPosition - param.xMax) * ...
   SpringContactCoefficient) + (DamperContactCoefficient * PistonVelocity);


// Lower Stroke Limit Contact Force [N] //
IF PistonPosition > 0 THEN
    SpringContactCoefficient := 0;
ELSIF PistonPosition <= 0 THEN
    SpringContactCoefficient := Spring;
END_IF;

IF PistonPosition > 0 THEN
    DamperContactCoefficient := 0;
ELSIF PistonPosition <= 0 THEN
    DamperContactCoefficient := Damper;
END_IF;

AbsPosition := ABS(PistonPosition);
F_Lower_Stroke_Limit := (SpringContactCoefficient * AbsPosition) + ...
   (DamperContactCoefficient * (-1) * PistonVelocity);


IF Main.Timer < 0 THEN
    y_tmp := 0;
ELSIF Main.Timer >= 0 THEN
    y_tmp := EXP((PistonVelocity * ...
       param.FrictionForceApproximationConstant) * 2.0);
    y_tmp :=  (y_tmp - 1.0) / (y_tmp + 1.0);
END_IF;

F_Stribeck_Friction := (((EXP(( -(PistonVelocity * y_tmp)) / ...
   param.StaticFrictionTimeConstant) * ABS(param.StaticFrictionForce)) + ...
   param.ColoumbFriction) * y_tmp) + (PistonVelocity * ...
   param.ViscousFrictionCoefficient);
```

```
// Piston Force Total [N]
F_Piston := ( (F_Hydraulic - F_Upper_Stroke_Limit + F_Lower_Stroke_Limit - ...
   F_Stribeck_Friction) );



//The below is not used anymore, since movement comes from unreal via ads:
//PistonAcceleration := (F_Piston - param.ExternalForce) / ...
   param.EffectiveMass;

//These are for running the hydraulic-mechanical system without any ...
   connection to Unreal.
//fbPistonVelocity(
//  ssMethodType := ssMethodType,
//  tCycle := param.tCycle,
//  In := PistonAcceleration,
//  Init := param.v0,
//  minOut := -param.vMax,
//  maxOut := param.vMax,
//  Out => PistonVelocity
//  );

//fbPistonPosition(
//  ssMethodType := ssMethodType,
//  tCycle := param.tCycle,
//  In := PistonVelocity,
//  Init := param.x0,
//  minOut := -5000,
//  maxOut := param.xMax+5000,
//  Out => PistonPosition
//  );
```

Listing B.6: HydraulicMotor (FB)

```
dpsum := (PressureIn - PressureOut);

Torque := ( dpsum * Dm * nhm * EXPT(10,(-6))) / (2 * par.pi);
```

Listing B.7: HPU (FB)

```
//Pump
fbPump(
MotorSpeedRPM := 413.2,
PumpDisplacement := 10.6E-6,
DeltaMaxPressure := 250.0E5,
PumpTorque => PumpTorque,
PumpFlowRate => PumpFlowRate
);

TankPressure := 1.2E5;              //[Pa]
SupplyPressure := 250E5;            //[Pa]

G_ActuationSystems.ReturnPressure := TankPressure;
G_ActuationSystems.SupplyPressure := SupplyPressure;
```

Listing B.8: TimeIntegrator (FB)

```
IF bEnableLimits THEN
```

```
    fOut := LIMIT(fMinVal, fOut + fIn*fStepTime, fMaxVal);
ELSE
    fOut := fOut + fIn*fStepTime;
END_IF
```

Listing B.9: DiscreteTimeIntegratos$_i nit(FB)$

```
CASE ssMethodType OF
    0:
        c_DiscreteTimeIntegrator_IC := 1;

    1:

        IF c_DiscreteTimeIntegrator_IC <> 0 THEN
            c_DiscreteTimeIntegrator_DS := Init;

            IF c_DiscreteTimeIntegrator_DS >= maxOut THEN
                c_DiscreteTimeIntegrator_DS := maxOut;
            ELSIF c_DiscreteTimeIntegrator_DS <= minOut THEN
                c_DiscreteTimeIntegrator_DS := minOut;
            END_IF;

        END_IF;

        IF c_DiscreteTimeIntegrator_DS >= maxOut THEN
            c_DiscreteTimeIntegrator_DS := maxOut;
        ELSIF c_DiscreteTimeIntegrator_DS <= minOut THEN
            c_DiscreteTimeIntegrator_DS := minOut;
        END_IF;


        Out := c_DiscreteTimeIntegrator_DS;

        c_DiscreteTimeIntegrator_IC := 0;
        c_DiscreteTimeIntegrator_DS := (tCycle * In) + ...
           c_DiscreteTimeIntegrator_DS;

        IF c_DiscreteTimeIntegrator_DS >= maxOut THEN
            c_DiscreteTimeIntegrator_DS := maxOut;
        ELSIF c_DiscreteTimeIntegrator_DS <= minOut THEN
            c_DiscreteTimeIntegrator_DS := minOut;
        END_IF;
END_CASE;
```

Listing B.10: ValveDynamics (FB)

```
IF (wn = 0.0) THEN
    RETURN;
END_IF

a := 1/EXPT(wn, 2.0);
b := dr/wn;

fAcceleration := (udcv_IN - 2*fbVelocity.fOut*b - fbPosition.fOut)/a;

fbVelocity(
    fIn := fAcceleration,
    fStepTime := fDeltaTime,
);

fbPosition(
```

```
    fIn := fbVelocity.fOut,
    fStepTime := fDeltaTime,
    bEnableLimits := TRUE,
    fMinVal := -1.0,
    fMaxVal := 1.0,
    fOut => DynamicValve
);
```

Listing B.11: OrificeEquation (FB)

```
rtb_Sum := (PressureIn - PressureOut)*1E-5;


IF rtb_Sum < 0.0 THEN
    y := -1.0;
ELSIF rtb_Sum > 0.0 THEN
    y := 1.0;
ELSE
    y := rtb_Sum;
END_IF;

lmin_to_m3s := 1.0/60000.0;


temp1 := ABS(rtb_Sum);
OrificeFlow := param.Kv * ValveOpening * y * SQRT(temp1) * lmin_to_m3s;
```

Listing B.12: Inputs

```
VAR_GLOBAL
    uDcvEnableMainJib : BOOL := TRUE;       //Enable/Disable manual ...
       control of MainJibCylinder valve
    uDcvMainJib : LREAL;
    uDcvEnableKnuckleJib : BOOL := TRUE;            //Enable/Disable ...
       manual control of BoomCylinder valve
    uDcvKnuckleJib : LREAL;
    uDcvEnableKingRotation : BOOL := TRUE;      //Enable/Disable manual ...
       control of KingRotationCylinder valve
    uDcvKingRotation : LREAL;
END_VAR
```

Listing B.13: Work (GVL)

```
VAR_GLOBAL
    //To Unreal engine
    FpistonMainBoom : LREAL;
    FpistonKnuckleJib : LREAL;
    FpistonKingRotation : LREAL;


    //From Unreal engine
    VpistonMainBoom : LREAL;
    VpistonKnuckleJib : LREAL;
    VpistonKingRotation : LREAL;

    XpistonMainBoom : LREAL;
    XpistonKnuckleJib : LREAL;
    XpistonKingRotation : LREAL;
END_VAR
```

Listing B.14: Param (GVL)

```
{attribute 'qualified_only'}
VAR_GLOBAL PERSISTENT
    pMainJibCylinder : ST_pValveControlledCylinder;              ...
       //Parameters for MainJibCylinder uses values from ...
       ST_pValveControlledCylinder
    pKnuckleJibCylinder : ST_pValveControlledCylinder;           ...
       //Parameters for KnuckleJib uses values from ...
       ST_pValveControlledCylinderv
    pKingRotationCylinder : ST_pValveControlledCylinder;         ...
       //Parameters for KingRotation uses values from ...
       ST_pValveControlledCylinderv
END_VAR
```

Listing B.15: Ramp (FB)

```
falling(CLK := bOn);
IF falling.Q THEN
    y := y0;
    t := 0;
END_IF

IF bOn THEN
    IF t < t0 THEN
        y := y0;
    ELSIF t < t1 THEN
        y := y0 + (y1-y0)*(t-t0)/(t1-t0);
    ELSIF t < t2 THEN
        y := y1;
    ELSIF t < t3 THEN
        y := y1 + (y3-y1)*(t-t2)/(t3-t2);
    ELSIF t < t4 THEN
        y := y3;
    ELSIF t < t5 THEN
        y := y3 + (y0-y3)*(t-t4)/(t5-t4);
    ELSE
        y := y0;
    END_IF

    t := t + Steptime;
END_IF
```

Listing B.16: $\mathrm{ST}_p ValveControlledCylinder(STRUCT)$

```
TYPE ST_pValveControlledCylinder :
STRUCT
    tCycle : LREAL := 1E-4;
    gravity : LREAL := 9.81;
    Kv : LREAL := 10 / SQRT(7);                          //[L/min / sqrt(bar)]
    DBore : LREAL := 280E-3;                             //[m]120E-3
    DRod : LREAL := 180E-3;                              //[m]80E-3
    xMax : LREAL := 3.07;                                //[m]
    pi : LREAL := 3.14159265359;                         //Value of pi
    VA0 : LREAL := 0.01;                                 //[m^3]
    VB0 : LREAL := 0.01;                                 //[m^3]
    FrictionConstant : LREAL := 10000;                   //[N/(m/s)]
    pA0 : LREAL := 0;
    pB0 : LREAL := 0;
    pAmax : LREAL :=250E5;                               //[pa]
    pBmax : LREAL :=250E5;                               //[pa]
```

```
    OilBulkModulus : LREAL := 12000E5;                    //[Pa]
    AdiatricAirConstant : LREAL := 1.4;                   //[-]
    AtmosphericPressure : LREAL := 1.01325E5;             //[Pa]
    VolumetricAirConstant : LREAL := 0.007;               //[%]
    ViscousFrictionCoefficient : LREAL := 1500;           //[kg/s]
    FrictionForceApproximationConstant : LREAL := 250;    //[-]
    ColoumbFriction : LREAL := 75;                        //[N] Force
    StaticFrictionForce : LREAL := 10000;                 //[N]
    StaticFrictionTimeConstant : LREAL := 0.02;           //[s/m]
    cyl_k_end : LREAL := 6E9;                             //[N/m] spring ...
       constant for spring at the stroke limits of a cylinder
    cyl_c_end : LREAL := 2E7;                             //[N*s/m] Viscous ...
       constant for damper at the stroke limits of a cylinder
END_STRUCT
END_TYPE
```

Listing B.17: Sign (FUN)

```
IF in > 0.0 THEN
    Sign := 1.0;
ELSE
    Sign := -1.0;
END_IF
```

Listing B.18: XboxControl

```
XboxControl.joyLeft := ...
   INT_TO_LREAL(XboxControl.controller.LeftJoystickX)/32767.0;
```

Listing B.19: Controller (STRUCT)

```
TYPE Controller :
STRUCT
    LeftJoystickX : INT;
    LeftJoystickY : INT;
    RightJoystickX : INT;
    RightJoystickY : INT;
    LeftTrigger : INT;
    RightTrigger : INT;
    LeftBumper : BOOL;
    RightBumper : BOOL;
    ButtonA : BOOL;
    ButtonB : BOOL;
    ButtonX : BOOL;
    ButtonY : BOOL;
    LeftThumb : BOOL;
    RightThumb : BOOL;
    Back : BOOL;
    Start : BOOL;
    DPadX : INT;
    DPadY : INT;
END_STRUCT
END_TYPE
```

Appendix C

Datasheet C

Unreal Engine Blueprint code

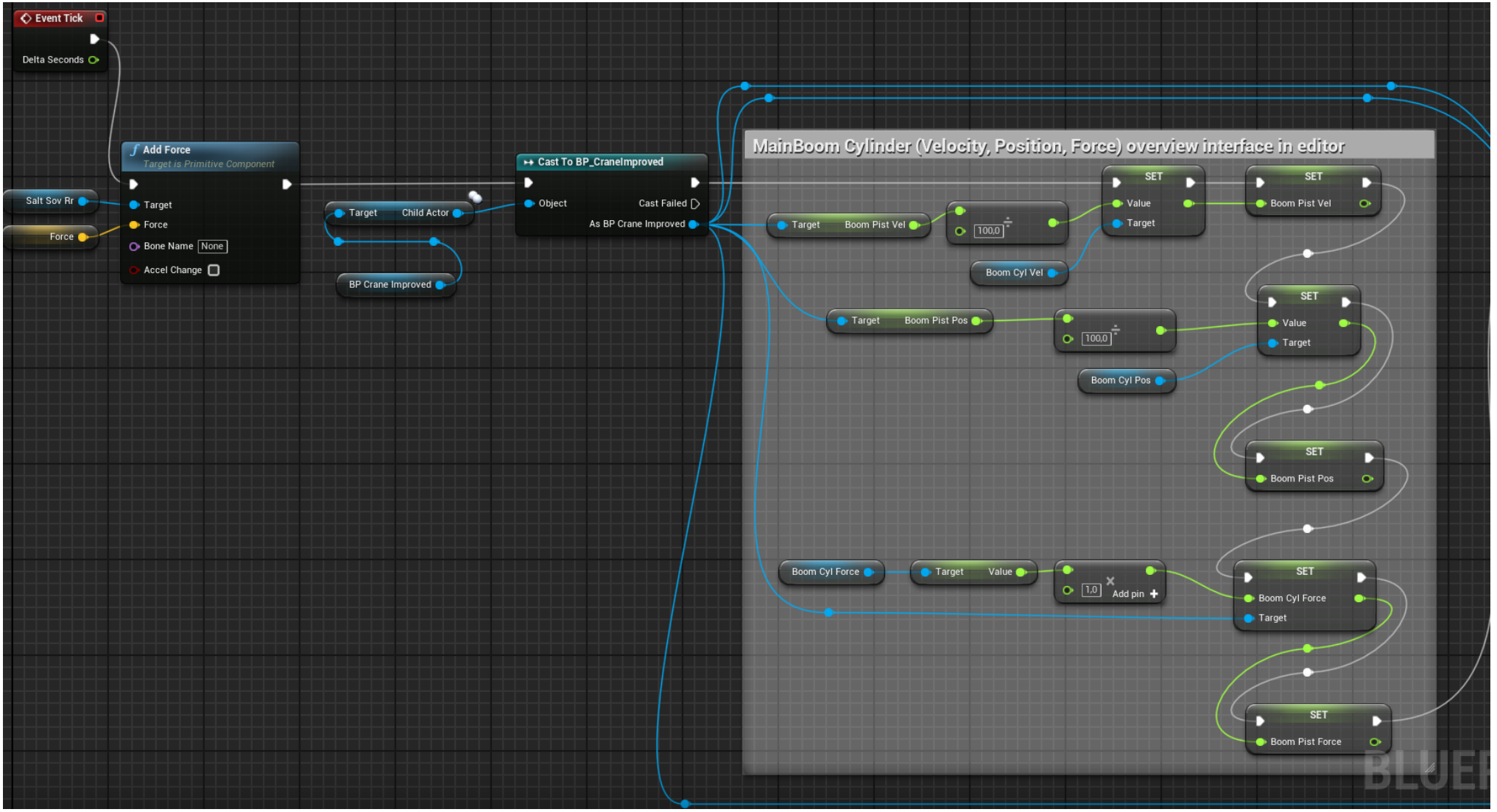

Event Tick
Delta Seconds
Add Force
Target is Primitive Component
Salt Sov Rr
Force
Target
Force
Bone Name None
Accel Change
Target Child Actor
BP Crane Improved
Cast To BP_CraneImproved
Object
Cast Failed
As BP Crane Improved
MainBoom Cylinder (Velocity, Position, Force) overview interface in editor
Target Boom Pist Vel
100,0
SET
Value
Target
SET
Boom Pist Vel
Boom Cyl Vel
Target Boom Pist Pos
100,0
SET
Value
Target
Boom Cyl Pos
SET
Boom Pist Pos
Boom Cyl Force
Target Value
1,0
Add pin
SET
Boom Cyl Force
Target
SET
Boom Pist Force

Figure C.1:

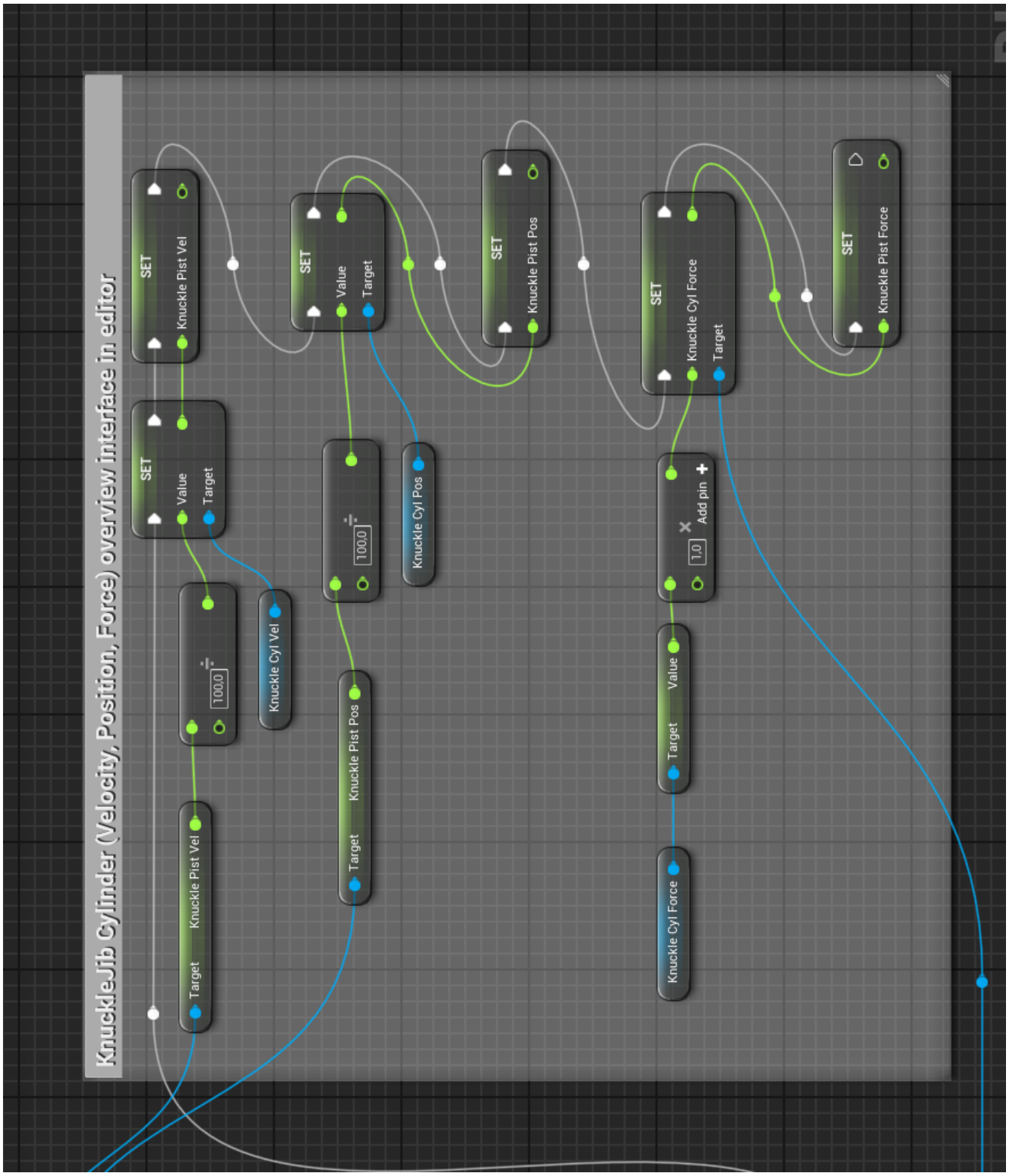

KnuckleJib Cylinder (Velocity, Position, Force) overview interface in editor
SET
Knuckle Pist Vel
Value
Target
100,0
Knuckle Cyl Vel
Knuckle Pist Pos
Knuckle Cyl Pos
Knuckle Cyl Force
1,0
Add pin

Figure C.2:

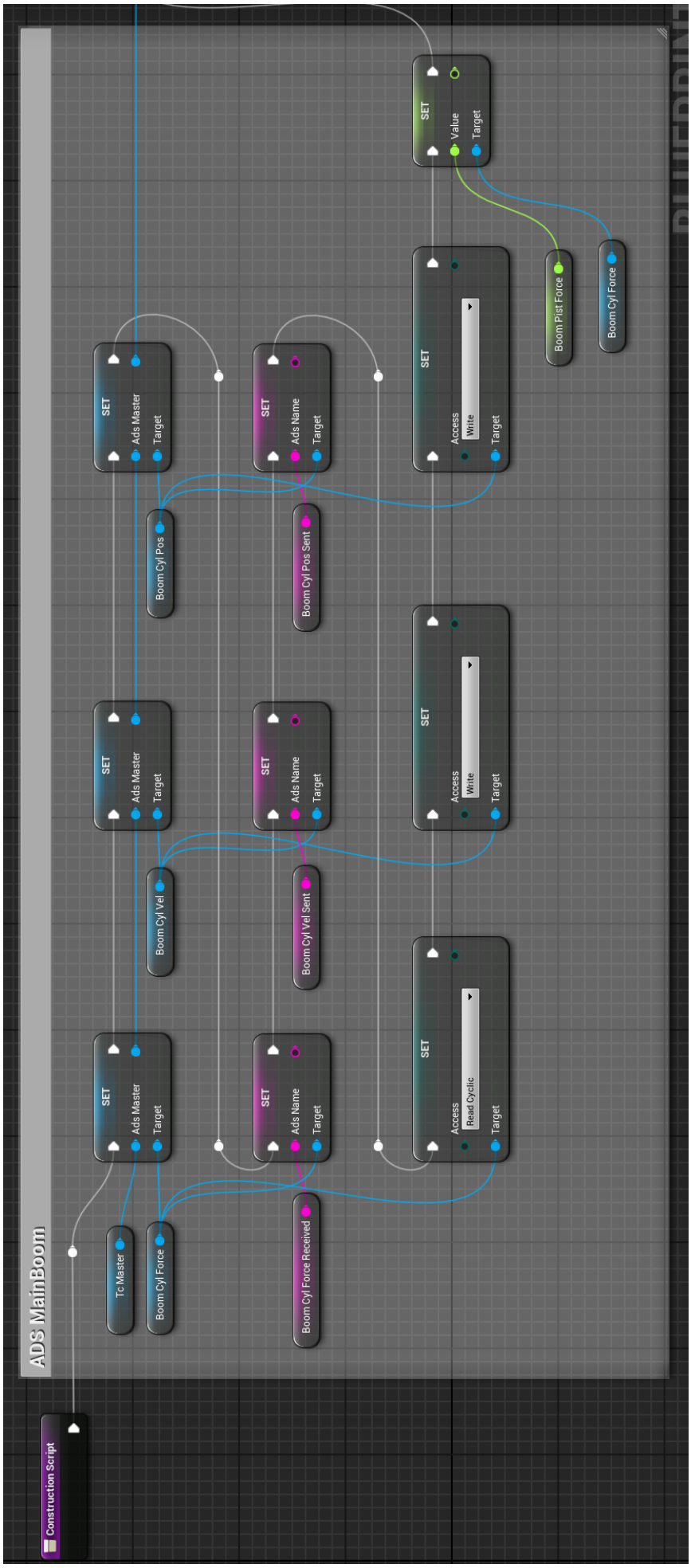

Construction Script
ADS MainBoom
Tc Master
Boom Cyl Force
Boom Cyl Force Received
SET
Ads Master
Target
Ads Name
Access
Write
Read Cyclic
Value
Boom Cyl Pos
Boom Cyl Pos Sent
Boom Cyl Vel
Boom Cyl Vel Sent
Boom Pist Force
Boom Cyl Force

Figure C.3:

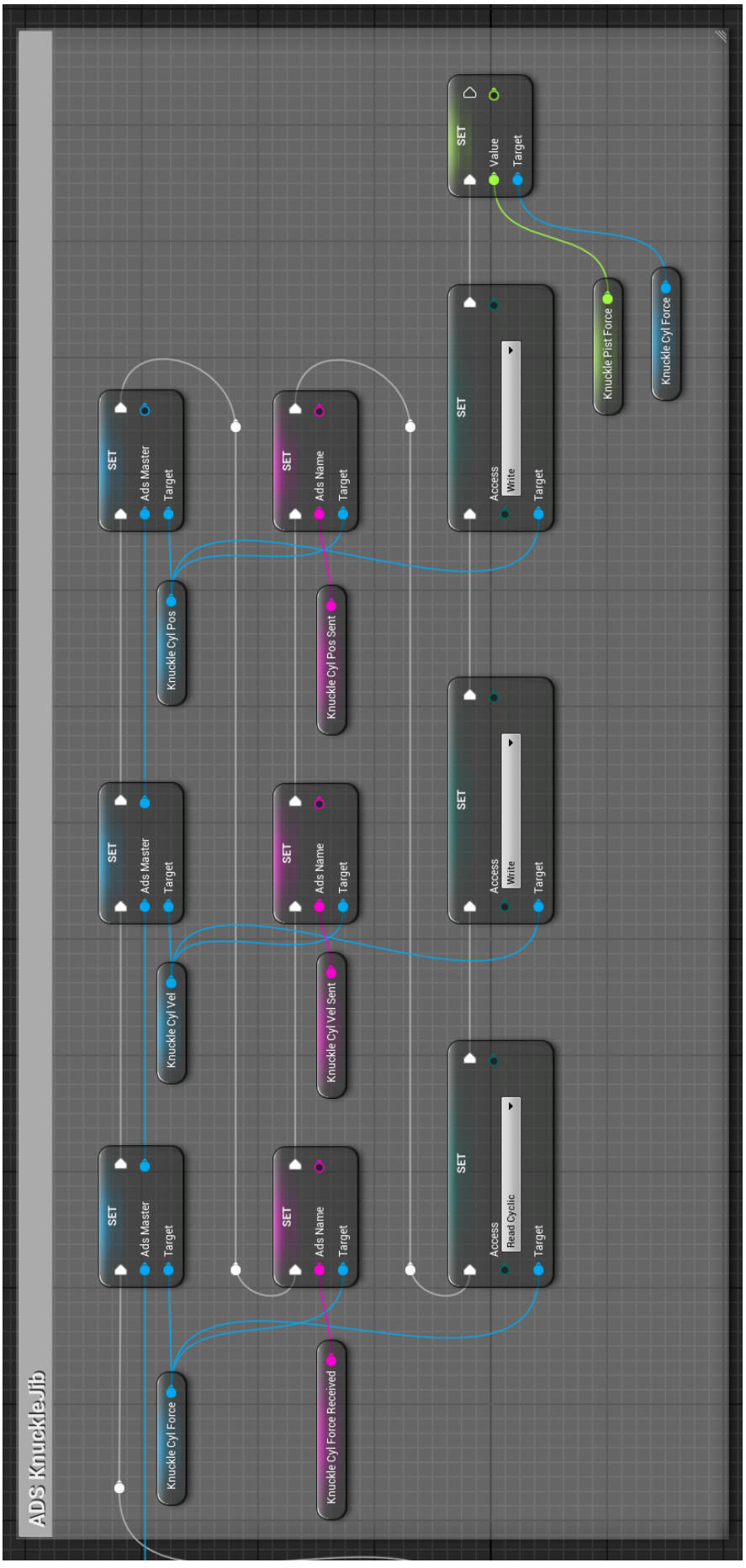

ADS KnuckleJib
SET
Ads Master
Target
Knuckle Cyl Pos
Knuckle Cyl Vel
Knuckle Cyl Force
Knuckle Cyl Force Received
SET
Ads Name
Target
Knuckle Cyl Pos Sent
Knuckle Cyl Vel Sent
SET
Access
Write
Read Cyclic
Target
SET
Value
Target
Knuckle Pist Force
Knuckle Cyl Force

Figure C.4:

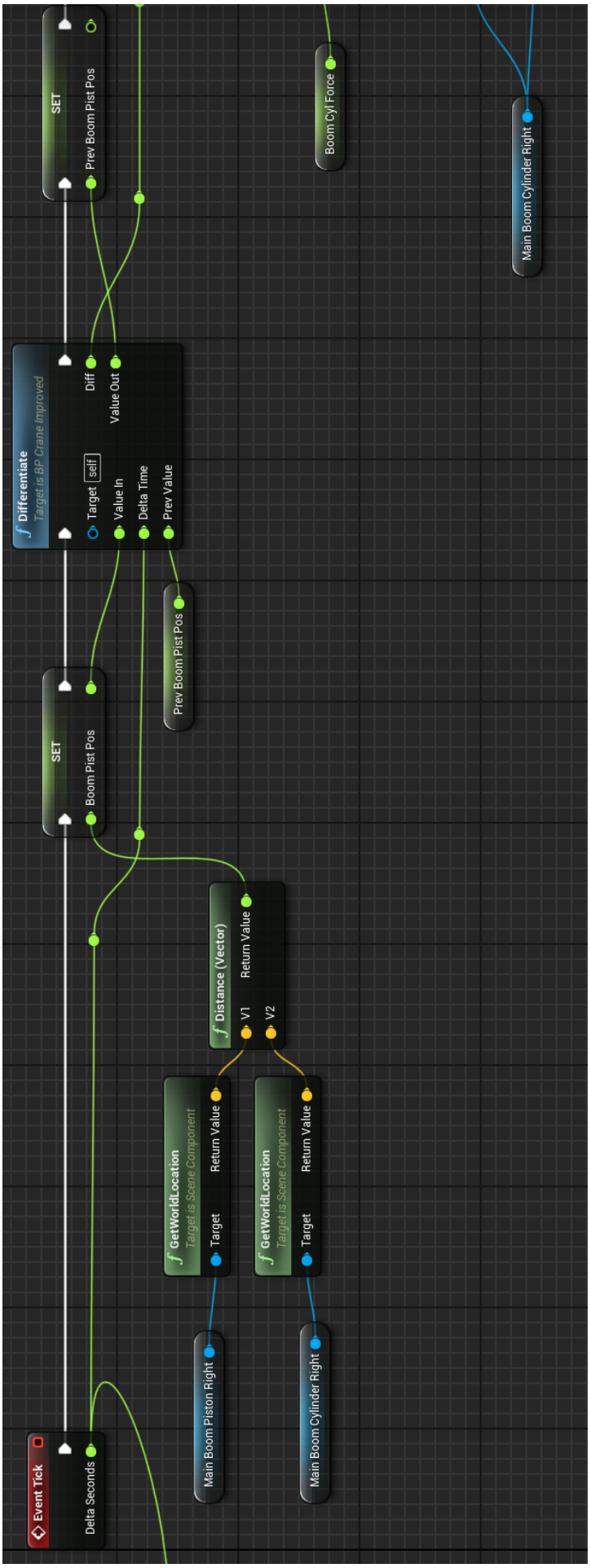

Event Tick
Delta Seconds
Main Boom Piston Right
Main Boom Cylinder Right
GetWorldLocation
Target is Scene Component
Target
Return Value
Distance (Vector)
V1
V2
Return Value
SET
Boom Pist Pos
Prev Boom Pist Pos
Differentiate
Target is BP Crane Improved
Target
self
Value In
Delta Time
Prev Value
Diff
Value Out
SET
Prev Boom Pist Pos
Boom Cyl Force
Main Boom Cylinder Right

Figure C.5:

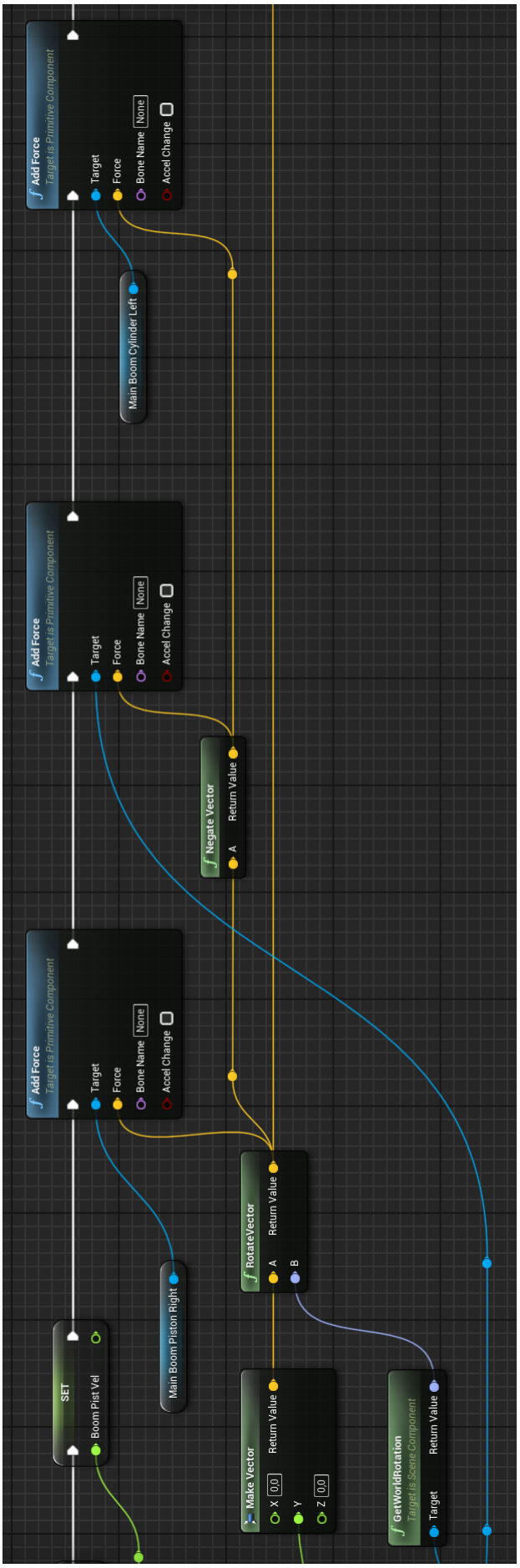

Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Main Boom Cylinder Left
Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Negate Vector
A
Return Value
Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
SET
Boom Pist Vel
Main Boom Piston Right
RotateVector
A
B
Return Value
Make Vector
X 0.0
Y
Z 0.0
Return Value
GetWorldRotation
Target is Scene Component
Target
Return Value

Figure C.6:

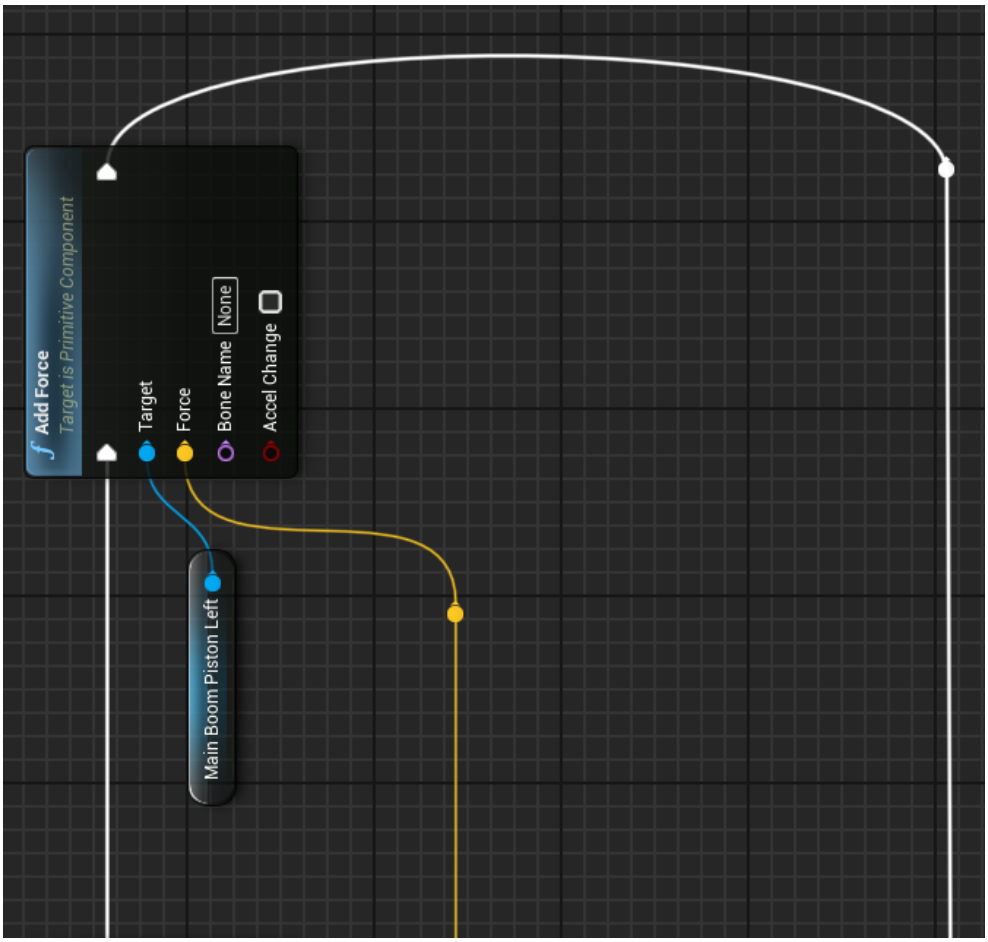

Add Force
Target is Primitive Component
Target
Force
Bone Name
None
Accel Change
Main Boom Piston Left

Figure C.7:

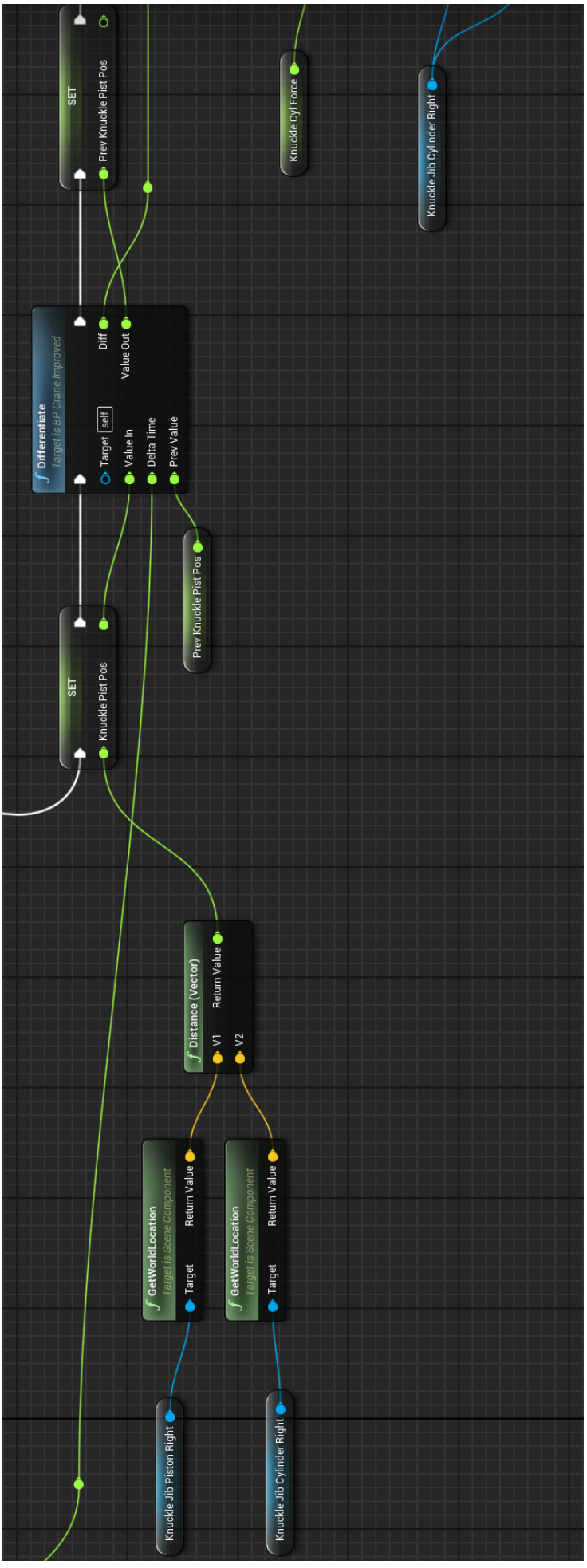

SET
Prev Knuckle Pist Pos
Knuckle Cyl Force
Knuckle Jib Cylinder Right
Differentiate
Target is BP Crane Improved
Target
self
Value In
Delta Time
Prev Value
Diff
Value Out
SET
Knuckle Pist Pos
Prev Knuckle Pist Pos
Distance (Vector)
V1
V2
Return Value
GetWorldLocation
Target is Scene Component
Target
Return Value
GetWorldLocation
Target is Scene Component
Target
Return Value
Knuckle Jib Piston Right
Knuckle Jib Cylinder Right

Figure C.8:

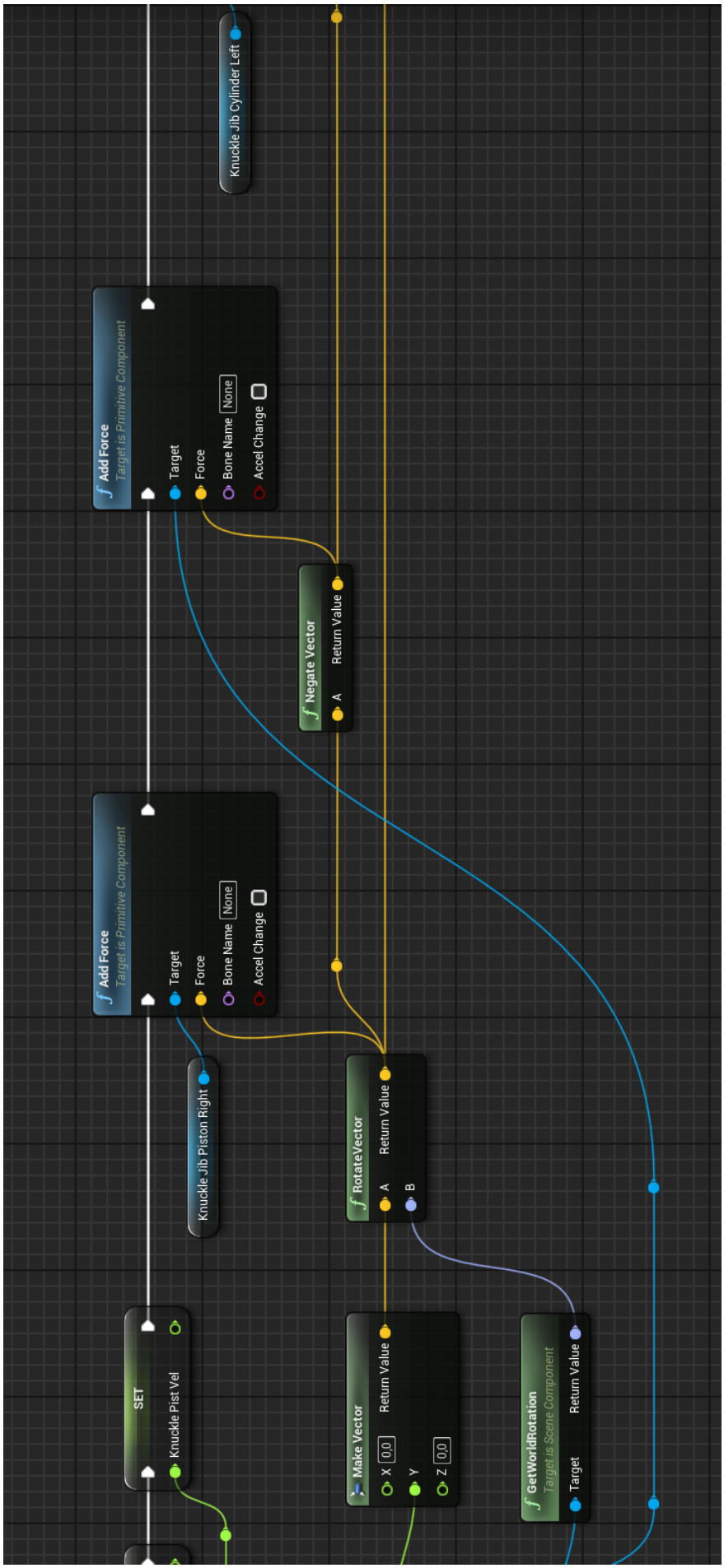

Knuckle Jib Cylinder Left
Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Negate Vector
A
Return Value
Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Knuckle Jib Piston Right
RotateVector
A
B
Return Value
SET
Knuckle Pist Vel
Make Vector
X 0.0
Y
Z 0.0
Return Value
GetWorldRotation
Target is Scene Component
Target
Return Value

Figure C.9:

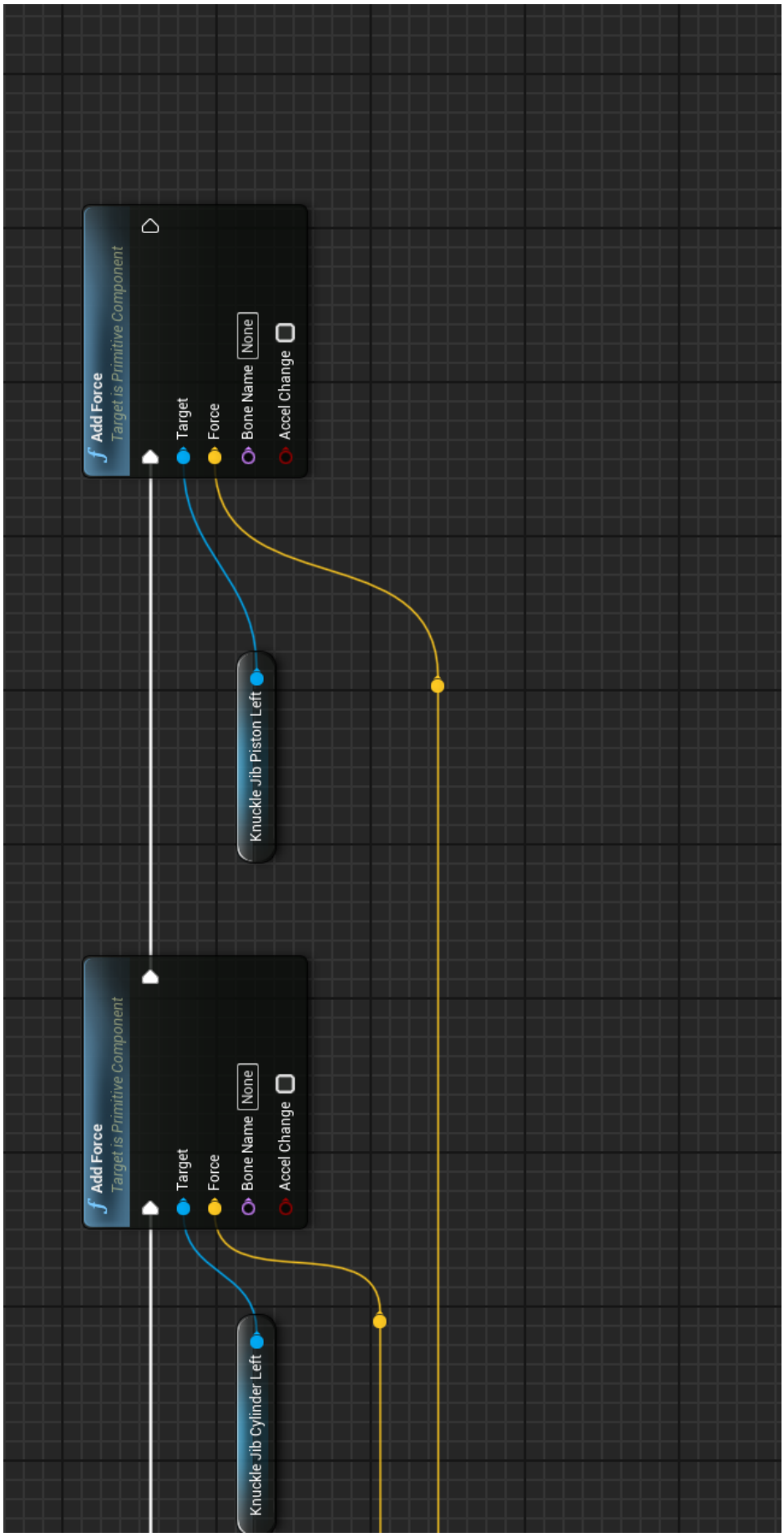
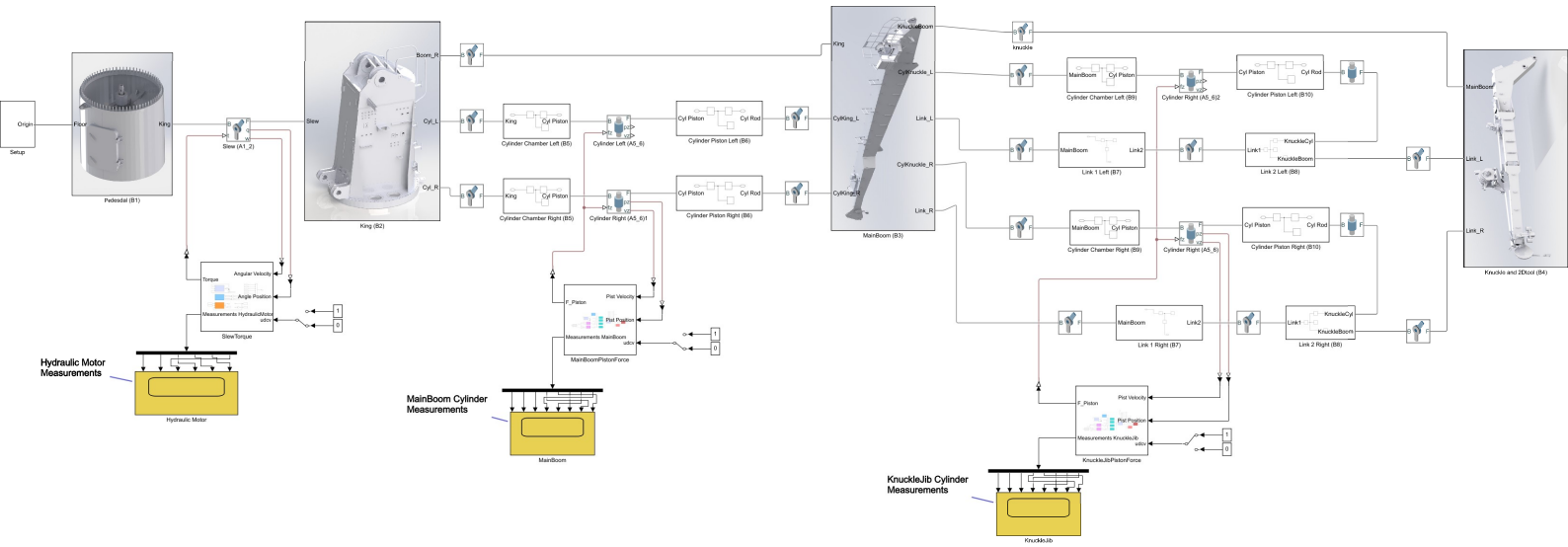

Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Knuckle Jib Piston Left
Add Force
Target is Primitive Component
Target
Force
Bone Name None
Accel Change
Knuckle Jib Cylinder Left

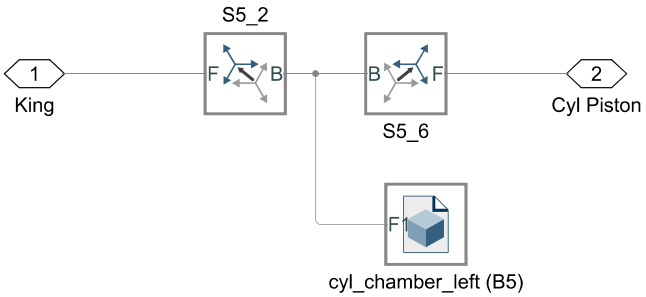
Figure C.10:

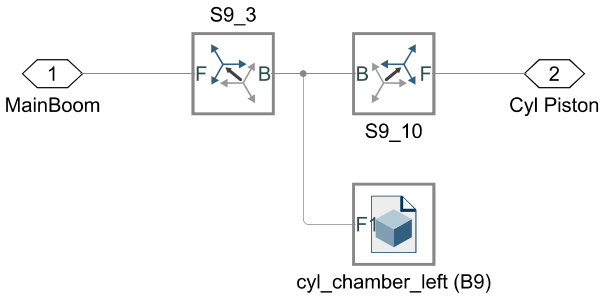
Appendix D

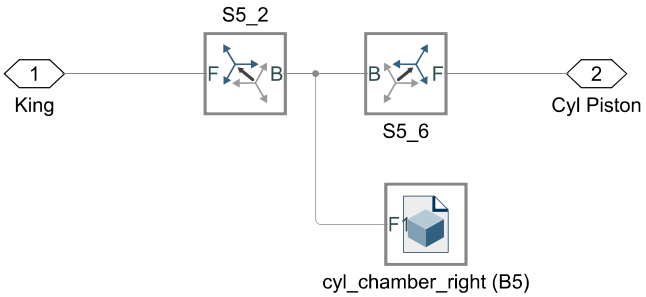
Datasheet D

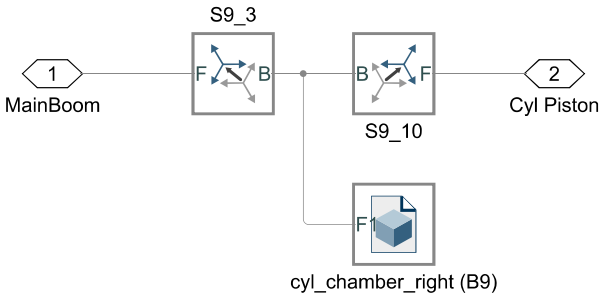
Simulink model

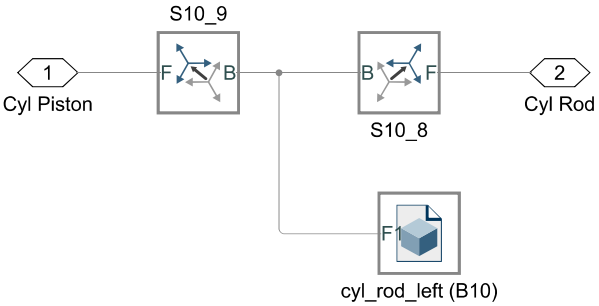

Origin
Setup
Floor
King
Pedestal (B1)
Slew (A1_2)
Slew
Boom_R
Cyl_L
Cyl_R
King (B2)
King
Cyl Piston
Cylinder Chamber Left (B5)
Cylinder Left (A5_6)
Cyl Piston
Cyl Rod
Cylinder Piston Left (B6)
King
Cyl Piston
Cylinder Chamber Right (B5)
Cylinder Right (A5_6)1
Cyl Piston
Cyl Rod
Cylinder Piston Right (B6)
KnuckleBoom
CylKing_L
CylKnuckle_L
Link_L
CylKnuckle_R
CylKing_R
Link_R
MainBoom (B3)
knuckle
MainBoom
Cyl Piston
Cylinder Chamber Left (B9)
Cylinder Right (A5_6)2
Cyl Piston
Cyl Rod
Cylinder Piston Left (B10)
MainBoom
Link2
Link 1 Left (B7)
Link1
KnuckleCyl
KnuckleBoom
Link 2 Left (B8)
MainBoom
Cyl Piston
Cylinder Chamber Right (B9)
Cylinder Right (A5_6)
Cyl Piston
Cyl Rod
Cylinder Piston Right (B10)
MainBoom
Link2
Link 1 Right (B7)
Link1
KnuckleCyl
KnuckleBoom
Link 2 Right (B8)
MainBoom
Link_L
Link_R
Knuckle and 2Dcol (B4)
Torque
Angular Velocity
Angle Position
Measurements HydraulicMotor
udcv
SlewTorque
Hydraulic Motor Measurements
Hydraulic Motor
F_Piston
Piston Velocity
Piston Position
Measurements MainBoom
udcv
MainBoomPistonForce
MainBoom Cylinder Measurements
MainBoom
F_Piston
Piston Velocity
Piston Position
Measurements KnuckleJib
udcv
KnuckleJibPistonForce
KnuckleJib Cylinder Measurements
KnuckleJib

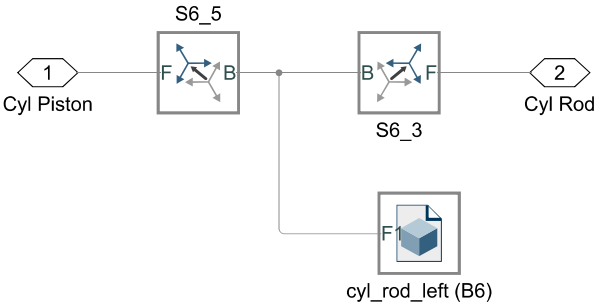

S5_2
1
King
F
B
B
F
S5_6
2
Cyl Piston
F1
cyl_chamber_left (B5)

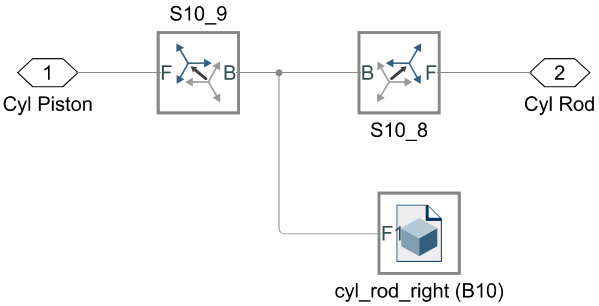

S9_3
1
MainBoom
F
B
B
F
S9_10
2
Cyl Piston
F1
cyl_chamber_left (B9)

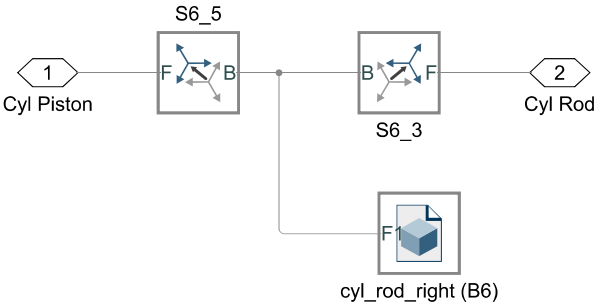

S5_2
1
King
F
B
B
F
S5_6
2
Cyl Piston
F1
cyl_chamber_right (B5)

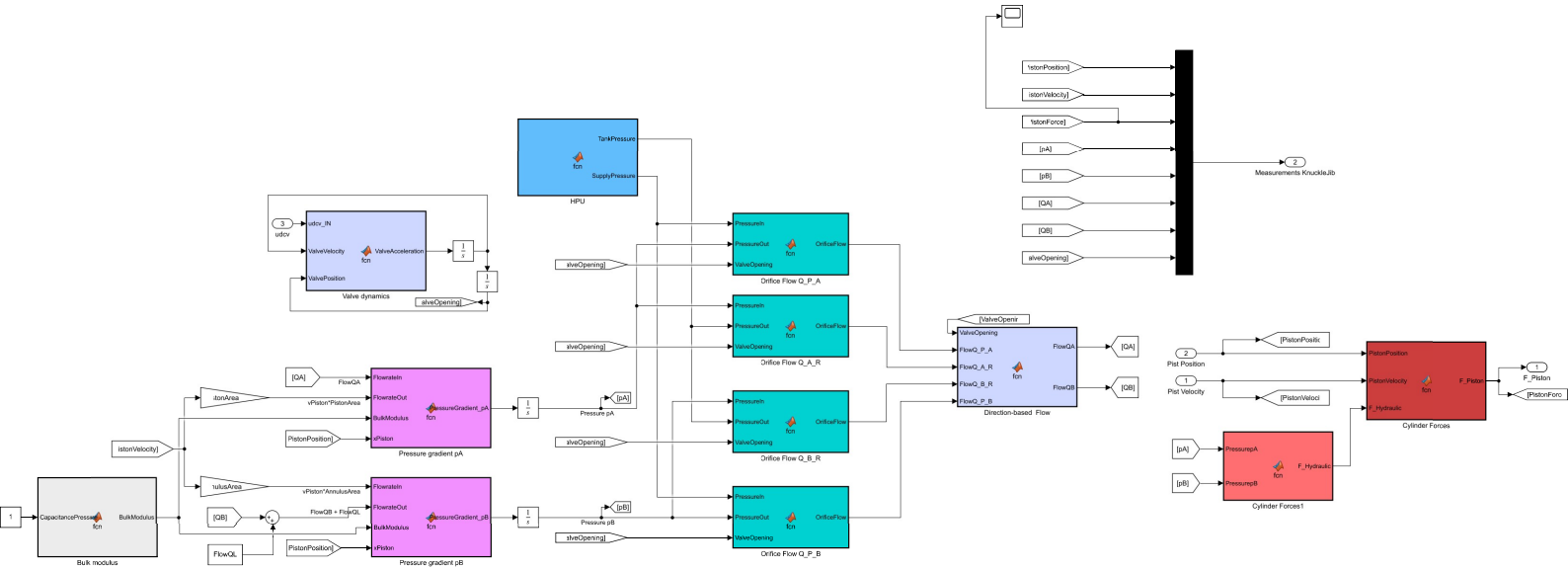

S9_3
1
MainBoom
F
B
B
F
S9_10
2
Cyl Piston
F1
cyl_chamber_right (B9)


S10_9
1
Cyl Piston
F
B
B
F
S10_8
2
Cyl Rod
F1
cyl_rod_left (B10)


S6_5
1
Cyl Piston
F
B
B
F
S6_3
2
Cyl Rod
F1
cyl_rod_left (B6)


S10_9
1
Cyl Piston
F
B
B
F
S10_8
2
Cyl Rod
F1
cyl_rod_right (B10)


S6_5
1
Cyl Piston
F
B
B
F
S6_3
2
Cyl Rod
F1
cyl_rod_right (B6)


HPU
TankPressure
SupplyPressure
Valve dynamics
udcv_IN
ValveVelocity
ValveAcceleration
ValvePosition
Orifice Flow Q_P_A
Orifice Flow Q_A_R
Orifice Flow Q_B_R
Orifice Flow Q_P_B
PressureIn
PressureOut
ValveOpening
OrificeFlow
Direction-based Flow
FlowQ_P_A
FlowQ_A_R
FlowQ_B_R
FlowQ_P_B
FlowQA
FlowQB
Pressure pA
Pressure pB
Pressure gradient pA
Pressure gradient pB
FlowrateIn
FlowrateOut
BulkModulus
xPiston
FlowQB + FlowQL
Bulk modulus
CapacitancePressure
Measurements KnuckleJib
Piston Position
Piston Velocity
Cylinder Forces
Cylinder Forces1
PistonPosition
PistonVelocity
F_Piston
F_Hydraulic
PressurepA
PressurepB
[QA]
[QB]
[pA]
[pB]
FlowQL

```
function BulkModulus = fcn(CapacitancePressure)
%AdiatricAirConstant = 1.4;                %[-]
%AtmosphericPressure = 1.01325E5;          %[Pa]
%OilBulkModulus = 12000E5;              %[Pa]
%VolumetricAirConstant = 0.007;            %[%]

BulkModulus = 12000E5 * CapacitancePressure;

%rtb_Add = AtmosphericPressure + CapacitancePressure;
%VariableBulkModulus = 1.0 / ((1.0 / OilBulkModulus) - (1.0 / ((((VolumetricAirContentOfOil - 1.0) / (EXPT(AtmosphericPressure / rtb
```

```
function F_Piston = fcn(PistonPosition,PistonVelocity,F_Hydraulic)
%Parameters
cyl_k_end = 6E9;                              %[N/m] spring constant for spring at the stroke limits of a cylinder
cyl_c_end = 2E7;                       %[N*s/m] Viscous constant for damper at the stroke limits of a cylinder
xMax = 2.32;                                  %[m]
ViscousFrictionCoefficient = 1500;           %[kg/s]
FrictionForceApproximationConstant = 250;    %[-]
ColoumbFriction = 75;                       %[N] Force
StaticFrictionForce = 10000;                %[N]
StaticFrictionTimeConstant = 0.02;             %[s/m]


%Upper Stroke Limit Contact Force [N]
if PistonPosition >= xMax
   SpringContactCoefficient = cyl_k_end;
else
   SpringContactCoefficient = 0;
end

if PistonPosition >= xMax
   DamperContactCoefficient = cyl_c_end;
else
   DamperContactCoefficient = 0;
end

F_Upper_Stroke_Limit = ((PistonPosition - xMax) * SpringContactCoefficient) + (DamperContactCoefficient * PistonVelocity);


%Lower Stroke Limit Contact Force [N]
if PistonPosition > 0
   SpringContactCoefficient = 0;
else
   SpringContactCoefficient = cyl_k_end;
end

if PistonPosition > 0
   DamperContactCoefficient = 0;
else
   DamperContactCoefficient = cyl_c_end;
end

AbsPosition = abs(PistonPosition);
F_Lower_Stroke_Limit = (SpringContactCoefficient * AbsPosition) + (DamperContactCoefficient * (-1) * PistonVelocity);


%Stribeck Friction Force [N]
y_tmp = exp(((PistonVelocity * FrictionForceApproximationConstant) * 2.0));
y_tmp = (y_tmp - 1.0) / (y_tmp + 1.0);
F_Stribeck_Friction = (((exp(( -(PistonVelocity * y_tmp)) / StaticFrictionTimeConstant) * abs(StaticFrictionForce)) + ColoumbFrictic


%Piston Force Total [N]
F_Piston = ( (F_Hydraulic - F_Upper_Stroke_Limit + F_Lower_Stroke_Limit - F_Stribeck_Friction) ) ;
```

```
function F_Hydraulic = fcn(PressurepA,PressurepB)
%DBore = 100E-3;                          %[m]120E-3
%DRod = 60E-3;                           %[m]80E-3
DBore = 220E-3;                           %[m]120E-3
DRod = 140E-3;                           %[m]80E-3
PistonArea = DBore^2 * (pi/4);
AnnulusArea = (DBore^2 - DRod^2) * (pi/4);

F_Hydraulic = PressurepA * PistonArea - PressurepB * AnnulusArea;
```

```
function [FlowQA,FlowQB] = fcn(ValveOpening,FlowQ_P_A,FlowQ_A_R,FlowQ_B_R,FlowQ_P_B)


if ValveOpening > 0
    FlowQA = FlowQ_P_A;
    FlowQB = FlowQ_B_R;
elseif ValveOpening < 0
    FlowQA = FlowQ_A_R;
    FlowQB = FlowQ_P_B;
else
    FlowQA = 0.0;
    FlowQB = 0.0;
end
```

```
function [TankPressure,SupplyPressure] = fcn()

TankPressure = 1.2E5;                  %[Pa]
SupplyPressure = 250E5;                   %[Pa]
```

```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

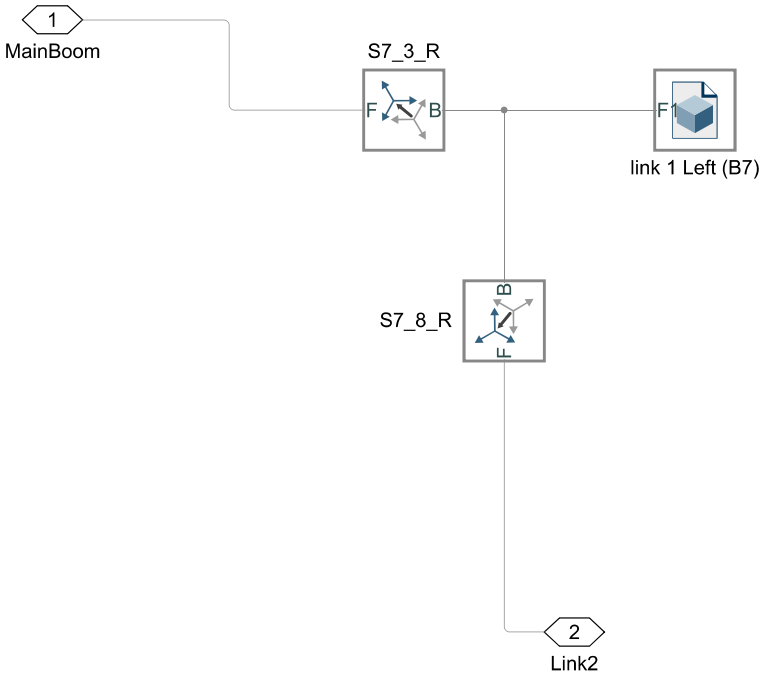
```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);

rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

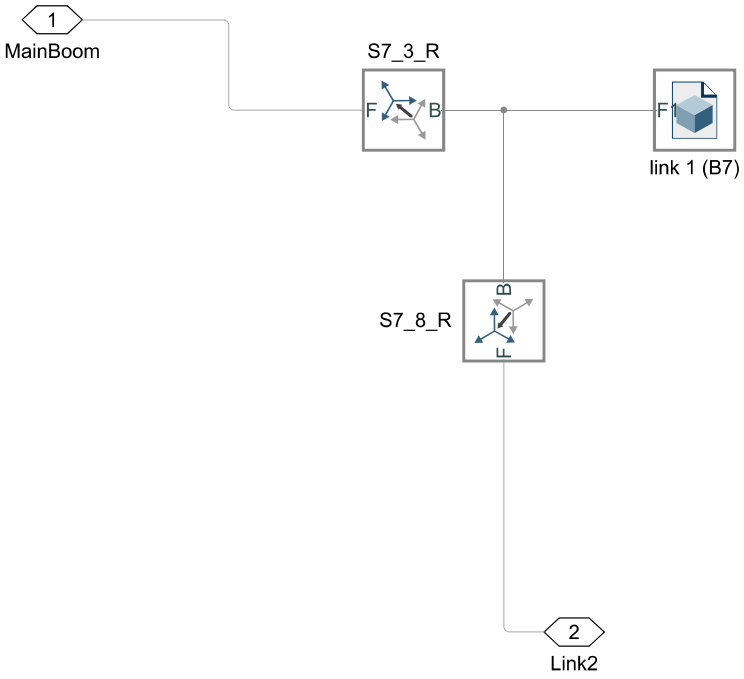
```
function OrificeFlow = fcn(PressureIn, PressureOut, ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;

if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;

temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

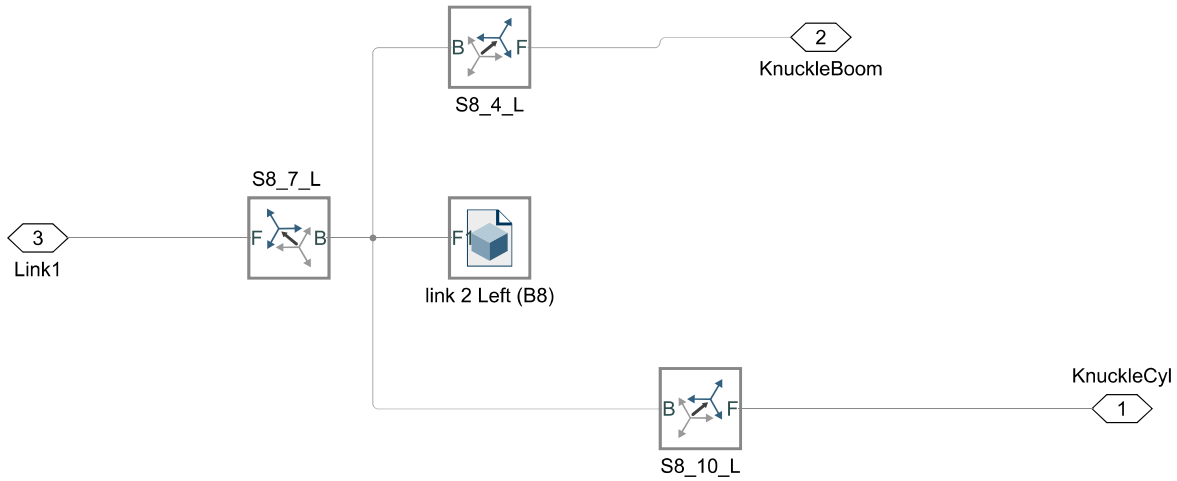
```
function PressureGradient_pA = fcn(FlowrateIn,FlowrateOut,BulkModulus,xPiston)
VA0 = 0.01;
%DBore = 100E-3;                          %[m]120E-3
%DRod = 60E-3;                           %[m]80E-3
DBore = 220E-3;                           %[m]120E-3
DRod = 140E-3;                           %[m]80E-3
PistonArea = DBore^2 * (pi/4);

TotalVolume = VA0 + ( xPiston * PistonArea );
PressureGradient_pA = (FlowrateIn - FlowrateOut) * (BulkModulus / TotalVolume);   %PressureGradient in [Pascal/s]
```

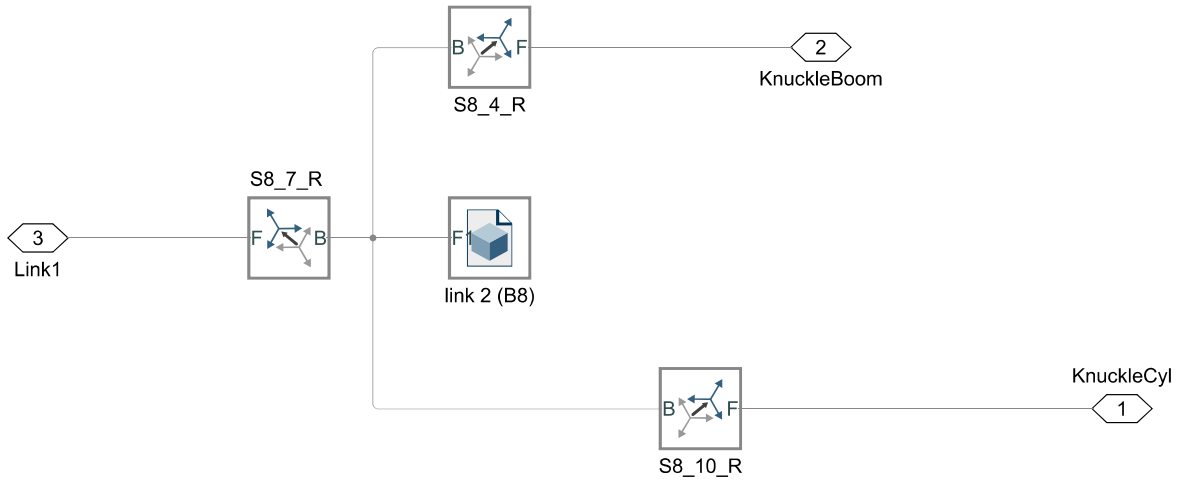
```
function PressureGradient_pB = fcn(FlowrateIn,FlowrateOut,BulkModulus,xPiston)
VB0 = 0.01;
xMax = 3;
%DBore = 100E-3;                          %[m]120E-3
%DRod = 60E-3;                           %[m]80E-3
DBore = 220E-3;                           %[m]120E-3
DRod = 140E-3;                           %[m]80E-3
AnnulusArea = (DBore^2 - DRod^2) * (pi/4);

TotalVolume = VB0 + ( (xMax - xPiston) * AnnulusArea );
PressureGradient_pB = (FlowrateIn - FlowrateOut) * (BulkModulus / TotalVolume);   %PressureGradient in [Pascal/s]
```

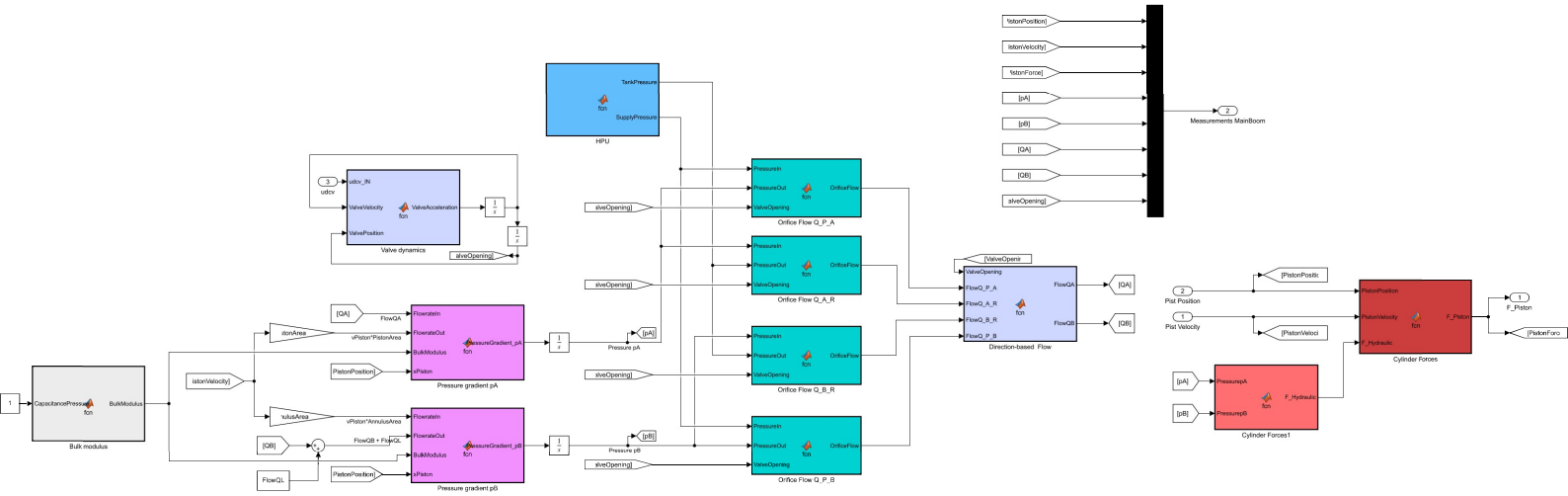
```
function ValveAcceleration = fcn(udcv_IN,ValveVelocity,ValvePosition)

wn = 30.0;
dr = 0.7;
a = 1/(wn^2);
b = dr/wn;

ValveAcceleration = (udcv_IN - 2*ValveVelocity*b - ValvePosition)/a;
```


1
MainBoom
S7_3_R
F
B
F1
link 1 Left (B7)
S7_8_R
B
F
2
Link2


1
MainBoom
S7_3_R
F
B
F1
link 1 (B7)
S7_8_R
F
B
2
Link2


B
F
2
KnuckleBoom
S8_4_L
S8_7_L
3
Link1
F
B
F
link 2 Left (B8)
B
F
KnuckleCyl
1
S8_10_L


B
F
S8_4_R
2
KnuckleBoom
S8_7_R
3
Link1
F
B
F1
link 2 (B8)
KnuckleCyl
B
F
S8_10_R
1


1
CapacitancePressure
fcn
BulkModulus
Bulk modulus
istonVelocity]
[QA]
FlowQA
itonArea
vPiston*PistonArea
FlowrateIn
FlowrateOut
BulkModulus
vPiston
PressureGradient_pA
fcn
PistonPosition]
Pressure gradient pA
1/s
[pA]
Pressure pA
*ulusArea
vPiston*AnnulusArea
[QB]
FlowQB + FlowQL
FlowQL
PressureGradient_pB
Pressure gradient pB
Pressure pB
[pB]
TankPressure
fcn
SupplyPressure
HPU
3
udcv
udcv_IN
ValveVelocity
ValveAcceleration
ValvePosition
Valve dynamics
aliveOpening]
PressureIn
PressureOut
ValveOpening
OrificeFlow
Orifice Flow Q_P_A
Orifice Flow Q_A_R
Orifice Flow Q_B_R
Orifice Flow Q_P_B
[ValveOpenir
FlowQ_P_A
FlowQ_A_R
FlowQ_B_R
FlowQ_P_B
FlowQA
FlowQB
Direction-based Flow
istonPosition]
istonVelocity]
istonForce]
[pA]
[pB]
[QA]
[QB]
aliveOpening]
2
Measurements MainBoom
2
Pist Position
1
Pist Velocity
[PistonPositic
[PistonVeloci
PistonPosition
PistonVelocity
F_Hydraulic
F_Piston
Cylinder Forces
1
F_Piston
[PistonForc
PressurepA
PressurepB
F_Hydraulic
Cylinder Forces1

```
function BulkModulus = fcn(CapacitancePressure)
%AdiatricAirConstant = 1.4;                %[-]
%AtmosphericPressure = 1.01325E5;          %[Pa]
%OilBulkModulus = 12000E5;              %[Pa]
%VolumetricAirConstant = 0.007;            %[%]

BulkModulus = 12000E5 * CapacitancePressure;

%rtb_Add = AtmosphericPressure + CapacitancePressure;
%VariableBulkModulus = 1.0 / ((1.0 / OilBulkModulus) - (1.0 / ((((VolumetricAirContentOfOil - 1.0) / (EXPT(AtmosphericPressure / rtb
```

```
function F_Piston = fcn(PistonPosition,PistonVelocity,F_Hydraulic)
%Parameters
cyl_k_end = 6E9;                              %[N/m] spring constant for spring at the stroke limits of a cylinder
cyl_c_end = 2E7;                      %[N*s/m] Viscous constant for damper at the stroke limits of a cylinder
xMax = 3.07;
ViscousFrictionCoefficient = 1500;          %[kg/s]
FrictionForceApproximationConstant = 250;    %[-]
ColoumbFriction = 75;                     %[N] Force
StaticFrictionForce = 10000;              %[N]
StaticFrictionTimeConstant = 0.02;           %[s/m]


%Upper Stroke Limit Contact Force [N]
if PistonPosition >= xMax
   SpringContactCoefficient = cyl_k_end;
else
   SpringContactCoefficient = 0;
end

if PistonPosition >= xMax
   DamperContactCoefficient = cyl_c_end;
else
   DamperContactCoefficient = 0;
end

F_Upper_Stroke_Limit = ((PistonPosition - xMax) * SpringContactCoefficient) + (DamperContactCoefficient * PistonVelocity);


%Lower Stroke Limit Contact Force [N]
if PistonPosition > 0
   SpringContactCoefficient = 0;
else
   SpringContactCoefficient = cyl_k_end;
end

if PistonPosition > 0
   DamperContactCoefficient = 0;
else
   DamperContactCoefficient = cyl_c_end;
end

AbsPosition = abs(PistonPosition);
F_Lower_Stroke_Limit = (SpringContactCoefficient * AbsPosition) + (DamperContactCoefficient * (-1) * PistonVelocity);


%Stribeck Friction Force [N]
y_tmp = exp(((PistonVelocity * FrictionForceApproximationConstant) * 2.0));
y_tmp = (y_tmp - 1.0) / (y_tmp + 1.0);
F_Stribeck_Friction = (((exp(( -(PistonVelocity * y_tmp)) / StaticFrictionTimeConstant) * abs(StaticFrictionForce)) + ColoumbFrictic


%Piston Force Total [N]
F_Piston = ( (F_Hydraulic - F_Upper_Stroke_Limit + F_Lower_Stroke_Limit - F_Stribeck_Friction) ) ;
```

```
function F_Hydraulic = fcn(PressurepA,PressurepB)
%DBore = 120E-3;                          %[m]120E-3
%DRod = 80E-3;                           %[m]80E-3
DBore = 280E-3;                           %[m]120E-3
DRod = 180E-3;                           %[m]80E-3

PistonArea = DBore^2 * (pi/4);
AnnulusArea = (DBore^2 - DRod^2) * (pi/4);

F_Hydraulic = PressurepA * PistonArea - PressurepB * AnnulusArea;
```

```
function [FlowQA,FlowQB] = fcn(ValveOpening,FlowQ_P_A,FlowQ_A_R,FlowQ_B_R,FlowQ_P_B)


if ValveOpening > 0
    FlowQA = FlowQ_P_A;
    FlowQB = FlowQ_B_R;
elseif ValveOpening < 0
    FlowQA = FlowQ_A_R;
    FlowQB = FlowQ_P_B;
else
    FlowQA = 0.0;
    FlowQB = 0.0;
end
```

```
function [TankPressure,SupplyPressure] = fcn()

TankPressure = 1.2E5;                  %[Pa]
SupplyPressure = 250E5;                   %[Pa]
```

```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

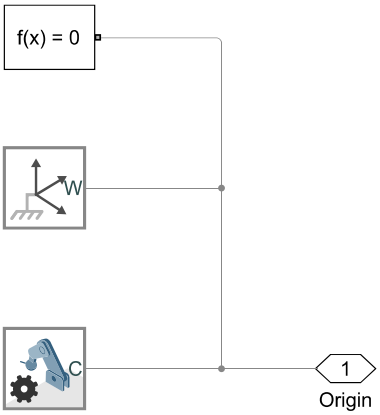
```
function OrificeFlow = fcn(PressureIn,PressureOut,ValveOpening)
Kv = 10 / sqrt(7);

rtb_Sum = (PressureIn - PressureOut)*1E-5;


if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;


temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

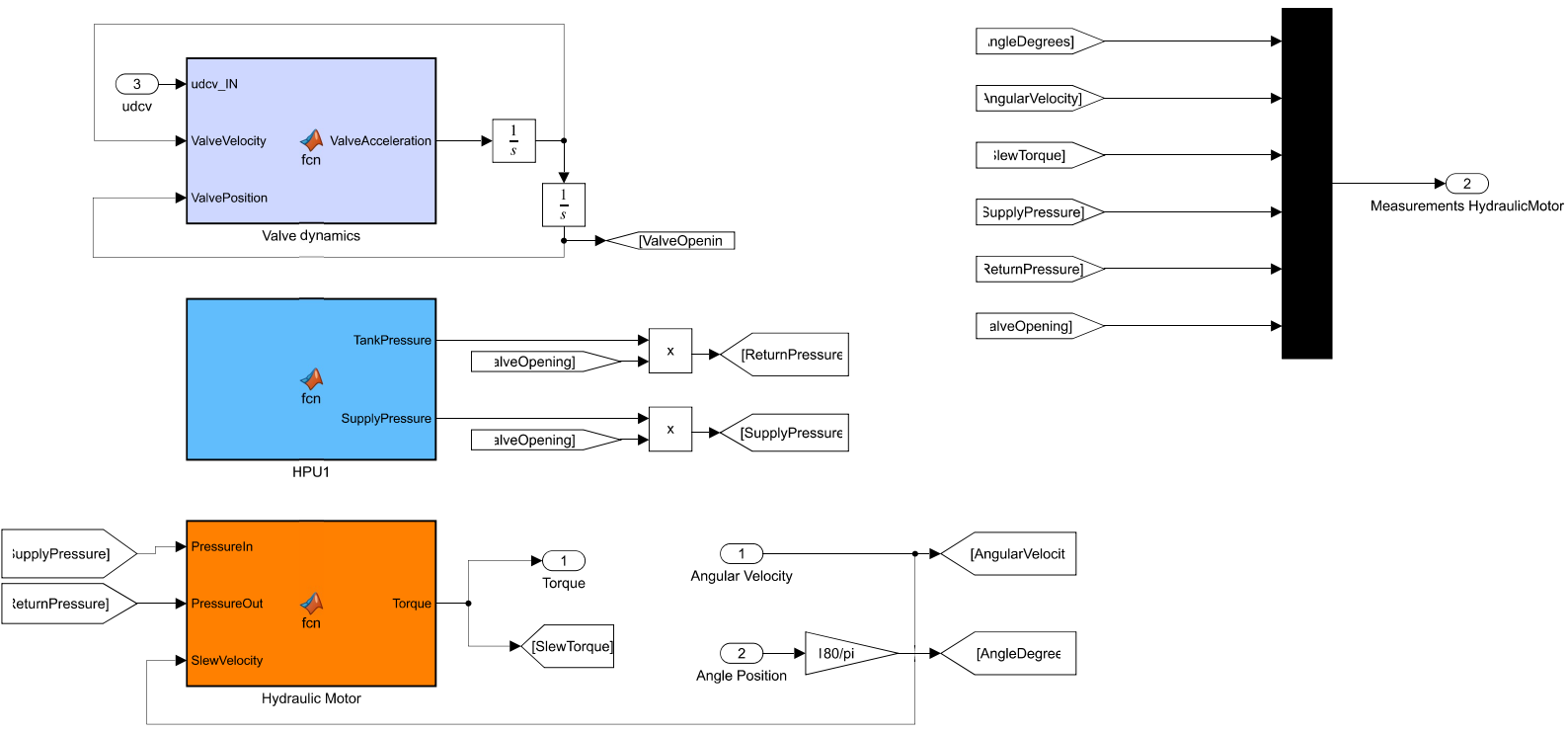
```
function OrificeFlow = fcn(PressureIn, PressureOut, ValveOpening)
Kv = 10 / sqrt(7);


rtb_Sum = (PressureIn - PressureOut)*1E-5;

if rtb_Sum < 0.0
    y = -1.0;
elseif rtb_Sum > 0.0
    y = 1.0;
else
    y = rtb_Sum;
end

lmin_to_m3s = 1.0/60000.0;

temp1 = abs(rtb_Sum);
OrificeFlow = Kv * ValveOpening * y * sqrt(temp1) * lmin_to_m3s;
```

```
function PressureGradient_pA = fcn(FlowrateIn,FlowrateOut,BulkModulus,xPiston)
VA0 = 0.01;
%DBore = 100E-3;                      %[m]120E-3
%DRod = 60E-3;                           %[m]80E-3
DBore = 280E-3;                           %[m]120E-3
DRod = 180E-3;                           %[m]80E-3
PistonArea = DBore^2 * (pi/4);

TotalVolume = VA0 + ( xPiston * PistonArea );
PressureGradient_pA = (FlowrateIn - FlowrateOut) * (BulkModulus / TotalVolume);   %PressureGradient in [Pascal/s]
```

```
function PressureGradient_pB = fcn(FlowrateIn,FlowrateOut,BulkModulus,xPiston)
VB0 = 0.01;
xMax = 3;
%DBore = 100E-3;                          %[m]120E-3
%DRod = 60E-3;                           %[m]80E-3
DBore = 280E-3;                           %[m]120E-3
DRod = 180E-3;                           %[m]80E-3
AnnulusArea = (DBore^2 - DRod^2) * (pi/4);

TotalVolume = VB0 + ( (xMax - xPiston) * AnnulusArea );
PressureGradient_pB = (FlowrateIn - FlowrateOut) * (BulkModulus / TotalVolume);   %PressureGradient in [Pascal/s]
```

```
function ValveAcceleration = fcn(udcv_IN,ValveVelocity,ValvePosition)

wn = 30.0;
dr = 0.7;
a = 1/(wn^2);
b = dr/wn;

ValveAcceleration = (udcv_IN - 2*ValveVelocity*b - ValvePosition)/a;
```


f(x) = 0
W
C
1
Origin


3
udcv
udcv_IN
ValveVelocity
ValveAcceleration
fcn
ValvePosition
Valve dynamics
1/s
1/s
[ValveOpenin
TankPressure
fcn
SupplyPressure
HPU1
alveOpening]
x
[ReturnPressure
alveOpening]
x
[SupplyPressure
ngleDegrees]
ngularVelocity]
lewTorque]
SupplyPressure]
ReturnPressure]
alveOpening]
2
Measurements HydraulicMotor
upplyPressure]
eturnPressure]
PressureIn
PressureOut
fcn
Torque
SlewVelocity
Hydraulic Motor
1
Torque
[SlewTorque]
1
Angular Velocity
2
Angle Position
180/pi
[AngularVelocit
[AngleDegree

```
function [TankPressure,SupplyPressure] = fcn()

TankPressure = 1.2E5;                  %[Pa]
SupplyPressure = 250E5;                   %[Pa]
```

```
function Torque = fcn(PressureIn,PressureOut,SlewVelocity)
Dm  = 189;                                  %[cm^3/Rev]
nhm = 0.9;                                   %[-]

dpsum = (PressureIn - PressureOut);          %[pa]
FrictionConstant = 150;

Torque = ( dpsum * Dm * nhm * 10^(-6)) / (2 * pi) - exp((SlewVelocity*FrictionConstant) * 2);
```

```
function ValveAcceleration = fcn(udcv_IN,ValveVelocity,ValvePosition)

wn = 30.0;
dr = 0.7;
a = 1/(wn^2);
b = dr/wn;

ValveAcceleration = (udcv_IN - 2*ValveVelocity*b - ValvePosition)/a;
```