UiA University of Agder

Master

Heave Compensated Timber Crane Model With Industrial Hardware

University of Agder Mechatronics Crane Model For Demonstration and Education

PER HENRIK MARIE HARDEBERG

SUPERVISORS
Morten Kjeld Ebbesen
Morten Hallquist Rudolfsen

University of Agder, 2024 Faculty of Engineering and Science Department of Engineering Sciences

Acknowledgments

First of all I am glad that I started with my studies at the University of Agder. My career started as a car mechanic, but my interest of mechanics and electronics pointed me in the direction of mechatronics. After the bachelors program, I was hooked, and could not say no to two more years. Now, looking back at the 5 years of mechatronics, including two exchange semesters, I am extremely grateful for all the people I have met. Interesting courses, good professors and fellow students is something I definitely will remember from these years.

When it comes to this project, I am really grateful for the opportunity to choose a project that fits my interest, by covering many aspects of my education. I want to thank Associate Professor Morten Kjeld Ebbesen and Morten Hallquist Rudolfsen for providing the project and being wonderful supervisors. Then I want to thank the University of Agder and staff in the lab for providing the necessary equipment, help and always be there for a discussion. I also want to thank other professors for helping me when I had questions in their field of interests, and a special thanks to Associate Professor Daniel Hagen and Professor Michael Hansen regarding this.

Last but not least I want to thank my all my fellow students for all the interesting talks, both professional and personal, but a special thanks to my fellow students Marcus Eide and Tollak Liland for the ability to share and discuss aspects of my project with them.

Per Henrik Hardeberg
22.05.2024
Grimstad

Abstract

This master thesis project aims to convert a 1:14 size hydraulic knuckle boom crane, into a instrumented demonstration model, for the mechatronics department at UiA. The demo model is meant for UiA to promote the mechatronics education for potential students, for example by bringing it to exhibits or schools. The crane has 4 degrees of freedom, 3 revolute joints and 1 prismatic joint, and is actuated by hydraulic cylinders. The hydraulic cylinders are controlled with small direction control valves (DCVs), that again are actuated by 5 [V] servo motors. The aim is to instrument this crane, with sensor on all DOFs. The control system includes manual control and end effector based in Cartesian coordinates, both using joystick. The top level controller is a Beckhoff industrial PC (IPC).

The project includes the design, choosing sensors, attaching them, and interface them with EtherCAT through Arduino and EasyCAT shields, to get the data to the Beckhoff IPC. 3 revolute joints are equipped with encoders, and the telescope distance is measured using a TOF distance sensor, and a light reflector inside the telescope.

The hardware design also includes a frame to attach the crane, including a designated workspace. The model is equipped with a tilting mechanism, to be able to disturb the crane orientation with respect to the frame, to simulate heave from a wave. The tilt has a orientation and position sensor (combined accelerometer and gyroscope), and active heave compensation is implemented in the demo model. This means that the end effector is striving to stay in the same position, even if the crane is disturbed.

The kinematics of the crane is derived and tested in Matlab Simulink. The kinematics of the crane is based on screw theory, and is generated to PLC code directly from Simulink, after the simulation shows satisfactory results.

The crane is actuator redundant, because there are 4 actuators with only 3 coordinates in Cartesian space to control. This makes the inverse kinematics analysis challenging, and is solved by removing the actuator redundancy. To remove the actuator redundancy, one of the actuators, the prismatic joint, is controlled independently in parallel, before the inverse kinematics solve the 3 other joint positions based on the actual joint position of all joints.

The model was successfully, designed, build and tested. Force, moments, pressure and flow was neglected in this project, mainly because the difficulty of measuring the flow and pressure in the small hoses of the hydraulic system. Using inverse kinematics to control end effector, and position controllers on the 3 revolute joints, the results shows under 2 [cm] error for most cases. Because the dynamics are neglected, the resultant control system is difficult to cover all crane configuration, for instance if the crane is fully extended, the joint error tends to be higher as expected.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background

In the landscape of engineering, the field of mechatronics is known as the field integrating mechanics, electronics, control theory and computer science. Mechatronics systems are widely used within manufacturing, automotive, healthcare, agriculture, oil and gas, among other things. Typical example of such systems can be robots and instrumented cranes.

1.2 Motivation

The Mechatronics department at the University of Agder (UiA) possesses a small hydraulic actuated knuckle boom crane, designed to be mounted on 1:14 sized remote-controlled trucks, such as the one in Figure 1.1. The project aims to convert this manual crane into an instrumented, heave-compensated, end effector controlled crane, mounted on a transportable frame. This demo model is meant to bring to exhibits, or schools to promote the mechatronics program at UIA, as well as using it in education.

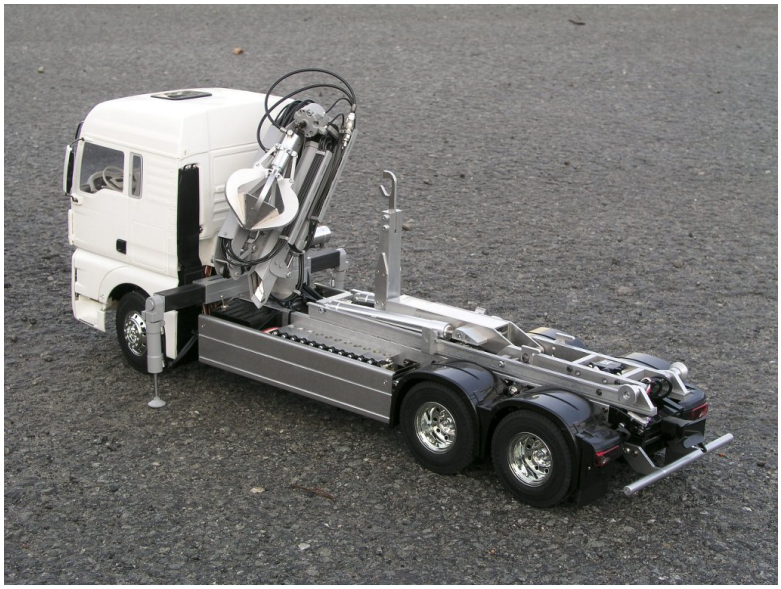

Figure 1.1: 1:14 Remote Control Truck [20]

1.3 System Description

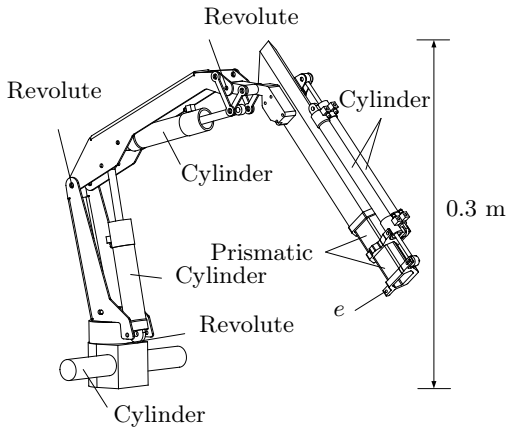

Figure 1.2: Crane Illustration

The crane that UiA possesses, includes a mini hydraulic system with a pump, hydraulic cylinders, hoses, directional control valves (DCVs), and servo motors for DCV control. Figure 1.2 shows the hydraulic actuators and joints of the crane. The crane has 3 revolute joints, each actuated by its corresponding hydraulic cylinder, and a prismatic joint formed by two cylinders connected in parallel, functioning as one telescope. This configuration gives the crane 4 degrees of freedom (DOF): 3 revolute and 1 prismatic.

1.4 Project Scope

For this project, a timber claw is attached to the point marked as e in Figure 1.2. The systems actuation redundancy, having 4 actuated DOF, but 3 coordinates at point e, presents a challenge.

Actuation redundancy is often used for dynamic control, such as for parallel manipulators [33] and Biarticularly-Actuated Robot Arms [27]. Even more relevant is the similar hydraulic crane presented in: [13]. In this case however, due to the simplicity of the hydraulic system, and the scope of the project it is decided to neglect the dynamics of this system. Making it an interesting challenge on how to solve the actuator redundancy problem, without using dynamics.

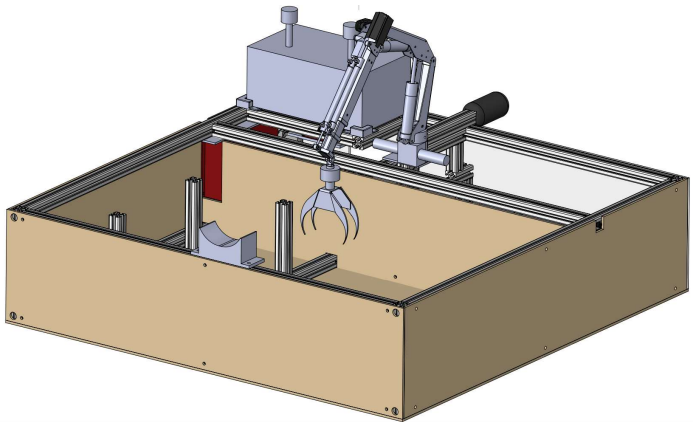

Figure 1.3: CAD Model Demo Model

Figure 1.3 shows the CAD model of the demo model. This report will present the design of the demo model and propose a solution to the actuator redundant problem, simulating the solution in Simulink, before implementing the solution to the hardware of the demo crane.

1.5 Objectives and Limitations

To be able to develop, build and test the demo model for UiA, there are many steps. To reach the goal, a set of concrete objectives are listed, together with some limitations.

Objectives

- Measure the crane parts, reverse engineer the crane and create CAD models
- Design and build a frame to attach the crane, hydraulic system, and controllers
- Chose appropriate sized sensors for feedback signal on each joint
- Design and build sensor attachment for each joint
- Forward kinematic analysis of the crane
- Inverse kinematic analysis of the crane
- Propose a solution to the actuation redundancy problem
- Interface the sensors and actuators with Beckhoff Industrial PC (IPC)
- Build a control box with joysticks for manual control of the crane
- Control the crane joint based using Beckhoff IPC
- Solve the hydraulic redundancy challenge and control the crane end-effector based in Cartesian coordinates using Beckhoff IPC
- Introduce safety stop before end of actuators
- Introduce Heave Compensation and a manual single axis disturbance to the crane

Limitations

- Simple hydraulic explanation
- No bearing cause slack in joints
- No load dynamics/no forces considered
- Simple joint based closed loop controller
- Simple path planning

1.6 Methods and Project Execution

Figure 1.4: Hardware Implementation

This section outline the different phases, to reach the project objectives. Because this project

is wide, including many different theoretical aspects, as well as the physical implementation and testing, all aspect are not covered in depth. This would cause this project to be to large, to be able to accomplish in one semester, as well as a long report. An example of tools used to make the project possible to finish, is pre made functions to solve the kinematics. This is explained in further detail in the multibody dynamics chapter. The complete demo model is presented in Figure 1.4.

In completing this project and writing the report, some inspiration from ChatGPT [24] is used. Specifically, ChatGPT assisted with some programming syntax, generating synonyms, and rewrite some sentences. However, it is not used to directly write or rewrite entire programs, paragraphs, or chapters.

For version control, GitHub is used through the project in a private repository. However, a new public repository called UIADemoCrane [26] is created to share the important files such as Simulink models, all Software and CAD model.

To be able to reach the set of objectives, the objectives are separated in to six main tasks. These task are assigned to a timeline as presented in Figure 1.5. Each task is separated into sub tasks. Trello [30] is used for the project management, where each sub task can be given a more accurate time slot if necessary, and it is easy to keep track of the task and their status. Each task is presented in more detail including sub tasks.

ID	Task Name	Start	Finish	Duration
1	Preparation	02.01.2024	02.02.2024	4,8w
2	Design	01.02.2024	29.02.2024	4,2w
3	Kinematics	15.01.2024	12.03.2024	8,4w
4	Build	01.03.2024	28.03.2024	4w
5	Implement and Test	25.03.2024	15.04.2024	3,2w
6	Report	15.04.2024	30.05.2024	6,8w

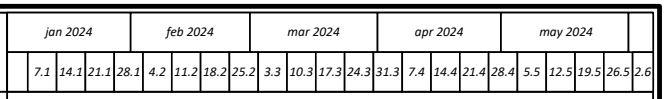

Figure 1.5: Gantt Diagram Project Execution

Preparation

- Choose sensors and other components with long shipping and order these (Priority)
- Write introduction of the report with clear objectives
- Learn the required hardware such as sensors and controllers
- Learn the required software for all controllers and shields
- Measure parts, and make CAD of crane

Design

- CAD Design frame including all components and crane
- Electrical design
- 3D Print all custom parts

Kinematics

- Read literature, talk to supervisors and decide type of kinematics
- Describe and simulate the forward and inverse kinematics for the crane

- Find a solution to the actuator redundancy

Building

- Check all 3D printed parts, and all hardware. Order missing parts (Priority)
- Build the complete crane and frame model
- Create wire harness with proper connectors and crimps
- Build Control Box

Implement and Test

- Establish Communication Between all Controllers and sensors
- Troubleshoot errors
- Implement Manual Mode
- Implement Cartesian Control
- Implement Heave Compensation
- Implement Safety Stop

Report

- Collect results
- Write report

1.7 Report Structure

The chapters are separated to try to make an easy step by step understanding of the complete demo model.

Introduction

This chapter starts to present the project background and motivation, before giving a brief system description. Then the project scope is presented followed by objectives, limitation, and methods. The last part of the introduction presents the report structure.

Multibody Dynamics

The multibody dynamics chapter will include the kinematics of the crane. This chapter will derive the forward kinematics as well as the inverse kinematics. The kinematics will then be used to connect the relationship between the position of each joint, and the position of the end effector of the crane. This chapter will also look closer into actuator redundancy, and propose a solution to remove the actuator redundancy.

Design

This chapter will present the reverse engineering of the given crane parts. This part will also describe the design of frame, sensor attachments and extraction of dimensions for further calculations.

Hardware

The hardware chapter will explain what kind of hardware is used, how it is working, and how it is connected together. The hardware chapter will also present a brief explanation of the hydraulic system and how the actuators work.

Control

The control theory chapter will discuss different control methods and how the kinematic analyses are connected to actuation.

Software

The software chapter will explain the IPC program. This will include the explanation of inputs and outputs, as well as the logic behind different modes for the crane. This chapter will also include the software and configurations for the other controllers and shields interfacing with the IPC.

Outcome and Results

This chapter will present some pictures of the completed demo model, together with the results from testing. The manual tests will be open loop control of each actuator, and the end effector control will present the results following a Cartesian reference.

Discussion

The results are evaluated and discussed. In addition, some challenges from the project, as well as further work will be discussed in this chapter.

Conclusion

This section will conclude on how the objectives met the actual result from the project.

Appendix

Includes user guide for the demo model, as well as some code and pictures.

Chapter 2

Multibody Dynamics

This chapter presents the multibody dynamics of the considered knuckle boom crane. First all the parts and joints are introduced with the complete crane. Then the crane is simplified and represented as a robotic manipulator with 3 revolute joints and 1 prismatic joint. The revolute joint and the prismatic joint illustration is presented in Figure 2.1. The prismatic joint is sometimes refereed to as the translational joint.

A convenient way to describe the rotations and translations of a robotic manipulator is to use the Denavit Hartenberg (D-H) parameters. Jacques Denavit and Richard Hartenberg first presented this notation in 1955 [15]. Richard P. Paul wrote a book called "Robot Manipulators: Mathematics, Programming, and Control" [25] in 1981. This book present several example on how the D-H parameters can be used.

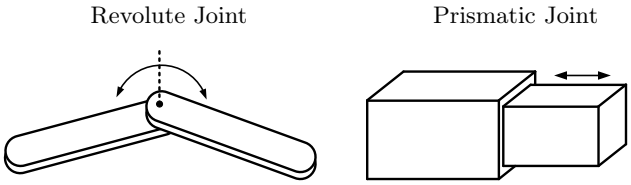

Figure 2.1: Different Joints Illustration

Another book presenting robot kinematics in a different way, is "Modern Robotics" by Kevin M. Lynch and Frank C. Park [18]. The book is using screw theory [34] in their kinematic calculations. This book is presenting several example and also links to a corresponding github code library [23], providing several pre made functions such as to calculate the forward, inverse, and velocity kinematics among other things. These functions are used for simulations in Matlab, before exported to PLC code, readable for the IPC, and implemented on the hardware. One of the major difference with the screw theory method, is that there is only two reference frames needed. The "ground" reference of the robotic manipulator, and the end-effector frame. The end effector or tool point, is the tip of the crane, where the load is attached. This will be illustrated clearly with Figures in the screw theory section. In comparison, the DH method requires one reference frame in each joint.

For a robotic manipulator the relationship between the joint space and the Cartesian space, is what we call the kinematics. Figure 2.2 illustrated a simple two dimentional robotic manipulater presented in the x_s , z_s plane. Based on the length of each body, and the angles q_2 and q_3, the vector $\mathbf{r} = \{x_s\ ,\ z_s\}$ can be found. This is forward kinematics. If the joint angles are calculated based on the Cartesian space coordinate $\{x_s\ ,\ z_s\}$, it is called inverse kinematics.

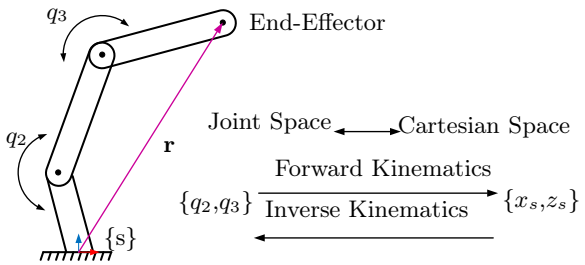

Figure 2.2: Forward and Inverse Kinematics

The forward kinematics for this robotic manipulator is presented with the famous Denavit-Hartenberg

representation first, then the screw theory method is used to represent the forward kinematics, inverse kinematics, and inverse velocity kinematics. The functions provided with the "Modern Robotics" book are used for simulations and validation of the kinematics. The Simulink Simulations in the end of this chapter are used to validate the forward and inverse kinematics. The actuator redundancy is be discussed, and a proposed solution is simulated in Simulink.

Because the crane is relatively small, all forces and dynamic analysis are neglected for this project. The main reason for this, is the small hydraulic component such as valves, and hoses. It is challenging to find suitable sensors for the flow and pressure.

2.1 Full Crane Simplified to Robotic Manipulator

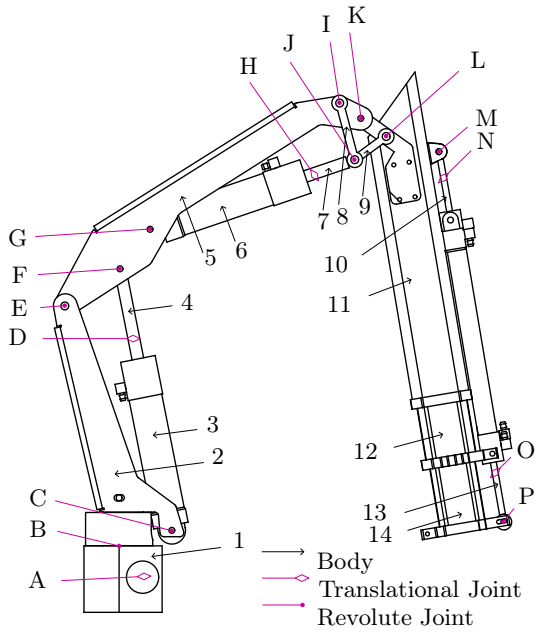

Figure 2.3: Total Bodies and Joints

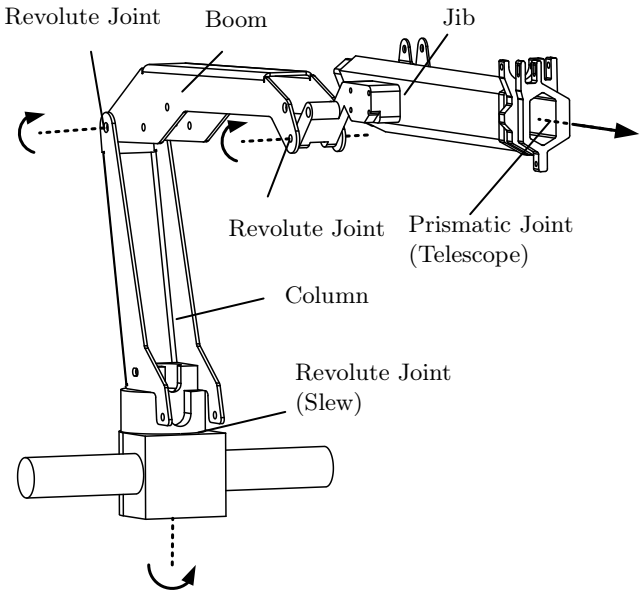

Figure 2.4: Simplified Crane

The knuckle-boom crane is illustrated in Figure 2.3. A body can be one part, or a set of parts, constrained by each other. Every part that is not fixed to another part is connected through a joint. For this crane, all the joints are marked in the Figure. The translational joints, joint A, D, H, N and O are actuators in the form of hydraulic cylinders.

The kinematics of the crane could be expressed using all the joints and bodies as they are expressed in Figure 2.3. However, since the magnetic encoders are located at the revolute joint B, E, and I, and this project is neglecting the dynamics of the crane it is appropriate to simplify the crane kinematics to only include revolute joints B, E, and K. For the closed loop controller, the relationship between joint I and K is mapped so the feedback signal can be converted from I to K. The complete jib/telescope assembly including body 10, 11, 12, 13 and 14, as well as joint M, N, O and P, are simplified to be only body 11 and 14 with one translational joint between them. The simplified crane is presented in Figure 2.4. Some common crane terms as slew, boom, jib, column, and telescope are also placed in the Figure.

2.2 Denavit–Hartenberg Representation

The Denavit–Hartenberg (DH) parameters is a convenient way to represent the translations between joints in a robotic manipulator [31]. The knuckle-boom crane can be simplified and represented in the same way as a robotic manipulator using the DH parameters.

Some rules are followed when writing the kinematic diagram. The first rule is that the z-axis have to point in the direction of the joint. In this case, for a revolute joint, the axis is pointing in the direction of the axis of rotation. For the translation joint, the axis is pointing parallel to the axis of motion. The other rule is that the x-axis is pointing in a direction to represent the displacement to next joint, if the z-axis is not able to represent [8]. With these two rules, the y-axis of each joint is not used, and the kinematics of the crane can be represented only using x and z.

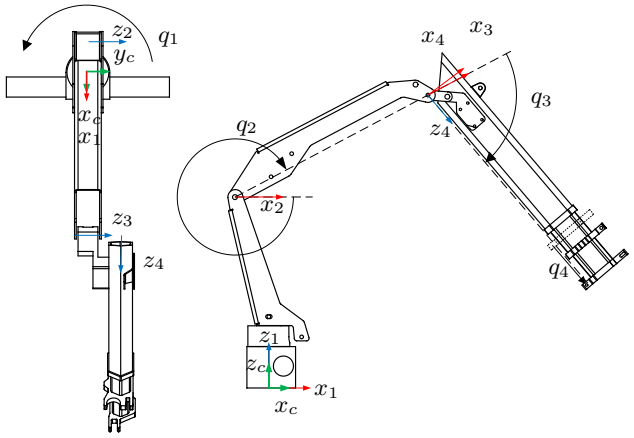

Figure 2.5: Kinematic Diagram for Angles

Figure 2.5 is representing the crane as a robotic manipulator with the joint coordinates and the angles attached.

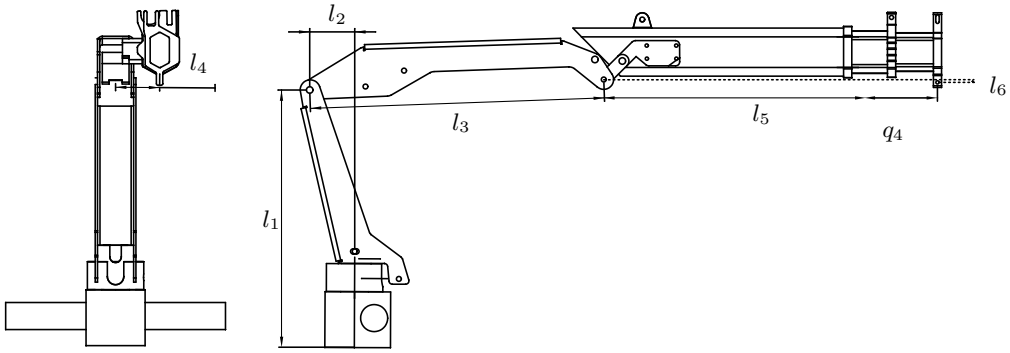

Figure 2.6: Kinematic Diagram

Figure 2.6 is presenting the length between the different joints. In this configuration q_1 , q_2 and q_3 = 0.

Link	θ_i	d_i	a_i	α_i
1	q_1	l_1	-l_2	$-\frac{\pi}{2}$
2	q_2	0	l_3	0
3	$q_3 - \frac{\pi}{2}$	l_4	0	$-\frac{\pi}{2}$
4	0	$l_5 + q_4$	$-l_6$	0

Table 2.1: DH Parameters

Lenght	**Value**
l_1	156.9 [mm]
l_2	27.45 [mm]
l_3	179.42 [mm]
l_4	26.86 [mm]
l_5	167.25 [mm]
l_6	1.75 [mm]

Table 2.2: Body Dimension

From Figure 2.6 the DH table is presented in Table 2.1. The dimensions are found from the Solidworks model and presented in Table 2.2.

From the DH parameter, the translation matrix of the crane can be derived [4]. For simplification purpose, sin is represented as s, and cos as c.

The translation matrix from joint z-1 to joint z is given as (In other words, the translation in z between the previous joint and the current one):

$$\mathbf{T}_i^z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.1}$$

The rotational matrix around z:

$$\mathbf{R}_i^z(\theta_i) = \begin{bmatrix} c\theta_i & -s\theta_i & 0 & 0 \\ s\theta_i & c\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.2}$$

The translation matrix from joint x to joint x+1 is given as (In other words, the translation in x between the current joint and the next one):

$$\mathbf{T}_i^x(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_n \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.3}$$

The rotational matrix around x:

$$\mathbf{R}^{\mathrm{x}}{}_i(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c\alpha_i & -s\alpha_i & 0 \\ 0 & s\alpha_i & c\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.4}$$

Multiply these 4 matrices together give the translation from one joint to the next joint.The combined translation matrix:

$$\mathbf{T}^{\mathrm{i+1}}{}_i(\theta_i, d_i, a_i, \alpha_i) = \begin{bmatrix} c\theta_i & -s\theta_i\, c\alpha_i & s\theta_i\, s\alpha_i & a_i\, c\theta_i \\ s\theta_i & c\theta_i\, c\alpha_i & -c\theta_i\, s\alpha_i & a_i\, s\theta_i \\ 0 & s\alpha_i & c\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.5}$$

This matrix can now be filled for each axis, with the values from the DH table in Table 2.1.

$$\mathbf{T}_0^1(q_1, l_1, -l_2, -\frac{\pi}{2}) = \begin{bmatrix} cq_1 & -sq_1\, c(-\frac{\pi}{2}) & sq_1\, s(-\frac{\pi}{2}) & -l_2\, cq_1 \\ sq_1 & cq_1\, c(-\frac{\pi}{2}) & -cq_1\, s(-\frac{\pi}{2}) & -l_2\, sq_1 \\ 0 & s(-\frac{\pi}{2}) & c(-\frac{\pi}{2}) & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.6}$$

$$\mathbf{T}_1^2(q_2, 0, l_3, 0) = \begin{bmatrix} cq_2 & -sq_2\, c0 & sq_2\, s0 & l_3\, cq_2 \\ sq_2 & cq_2\, c0 & -cq_2\, s0 & l_3\, sq_2 \\ 0 & s0 & c0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.7}$$

$$\mathbf{T}_2^3 = \begin{bmatrix} c(q_3 - \frac{\pi}{2}) & -s(q_3 - \frac{\pi}{2})\, c(-\frac{\pi}{2}) & s(q_3 - \frac{\pi}{2})\, s(-\frac{\pi}{2}) & 0\, c(q_3 - \frac{\pi}{2}) \\ s(q_3 - \frac{\pi}{2}) & c(q_3 - \frac{\pi}{2})\, c(-\frac{\pi}{2}) & -c(q_3 - \frac{\pi}{2})\, s(-\frac{\pi}{2}) & 0\, s(q_3 - \frac{\pi}{2}) \\ 0 & s(-\frac{\pi}{2}) & c(-\frac{\pi}{2}) & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.8}$$

$$\mathbf{T}_3^4(0, d_i, 0, 0) = \begin{bmatrix} c(0) & -s(0)\ c(0) & s(0)\ s(0) & (-l_6)\ c(0) \\ s(0) & c(0)\ c(0) & -c(0)\ s(0) & (-l_6)\ s(0) \\ (0) & s(0) & c(0) & (l_5 + q_4) \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.9}$$

The total transformation matrix is then:

$$\mathbf{T}_0^4 = \mathbf{T}_0^1 \mathbf{T}_1^2 \mathbf{T}_2^3 \mathbf{T}_3^4 \tag{2.10}$$

This transformation matrix is then a function of q_1, q_2, q_3 and q_4.

The transformation matrix for each joint, or for the complete robotic arm, is containing the rotation matrix as well as the translation vector. The following equation presents transformation matrix $\mathbf{T}_0^4$ that is showing the location of the rotation matrix $\mathbf{R}_0^4$ as well as the translation $\mathbf{Tr}_0^4$.

$$\mathbf{T}_0^4 = \left[\begin{array}{ccc|c} & \mathbf{R}_0^4 & & \mathbf{Tr}_0^4 \\ \hline 0 & 0 & 0 & 1 \end{array}\right] = \begin{bmatrix} c(q_1) & 0 & -s(q_1) & -l_2\ c(q_1) \\ s(q_1) & 0 & c(q_1) & -l_2\ s(q_1) \\ 0 & -1 & 0 & l_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} c(q_2) & -s(q_2) & 0 & l_3\ c(q_2) \\ s(q_2) & c(q_2) & 0 & l_3\ s(q_2) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$
$$\begin{bmatrix} c(q_3 - \frac{\pi}{2}) & 0 & -s(q_3 - \frac{\pi}{2}) & 0 \\ s(q_3 - \frac{\pi}{2}) & -c(q_3 - \frac{\pi}{2}) & 0 & 0 \\ 0 & -1 & 0 & l_4 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & -l_6 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & l_5 + q_4 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Because the total $\mathbf{T}_0^4$ is large to fit the page, the transformation part is presented as $\mathbf{Tr}_0^4 = [\ \mathrm{TrX}_0^4; \mathrm{TrY}_0^4; \mathrm{TrZ}_0^4]$:

$$\begin{aligned} TrX_0^4 &= l_6(c(q_1)s(q_2)s(q_3 - \frac{\pi}{2}) - c(q_1)c(q_2)c(q_3 - \frac{\pi}{2})) - l_2c(q_1) - l_4s(q_1) + l_3c(q_1)c(q_2) \\ &\quad - c(q_1)c(q_2)s(q_3 - \frac{\pi}{2})(l_5 + q_4) \\ TrY_0^4 &= l_6(s(q_1)s(q_2)s(q_3 - \frac{\pi}{2}) - c(q_2)s(q_1)c(q_3 - \frac{\pi}{2})) + l_4c(q_1) - l_2s(q_1) + l_3c(q_2)s(q_1) \\ &\quad - c(q_2)s(q_1)s(q_3 - \frac{\pi}{2})(l_5 + q_4) \\ TrZ_0^4 &= l_1 + l_6(c(q_2)s(q_3 - \frac{\pi}{2}) + c(q_3 - \frac{\pi}{2})s(q_2)) - l_3s(q_2) + s(q_2)s(q_3 - \frac{\pi}{2})(l_5 + q_4) \end{aligned}$$

Variable	**Value**
q_1	$-\pi/10$ [rad]
q_2	$-\pi/4$ [rad]
q_3	$\frac{\pi}{2}$ [rad]
q_4	100 [mm]

Table 2.3: Input Variables for Validation

To validate the transformation matrix, some angles presented in Table 2.3 are inserted in the transformation matrices, and the resultant transformation is calculated and plotted. The position of each joint is calculated using the transformation matrix from the origin to the given joint. For example, position of joint q_3 is found multiplying $T_0^1 T_0^2$.

The result position of all joints with input $q_1 - q_4 = 0$ is the green lines in Figure 2.7. This looks similar to the kinematic diagram in Figure 2.6, used to find the parameters to the DH-table. Next, the angles are given to the joints and the resultant crane visualization is the blue lines in Figure 2.7. The given input is given in Table 2.3. The calculations are presented in appendix B.9.

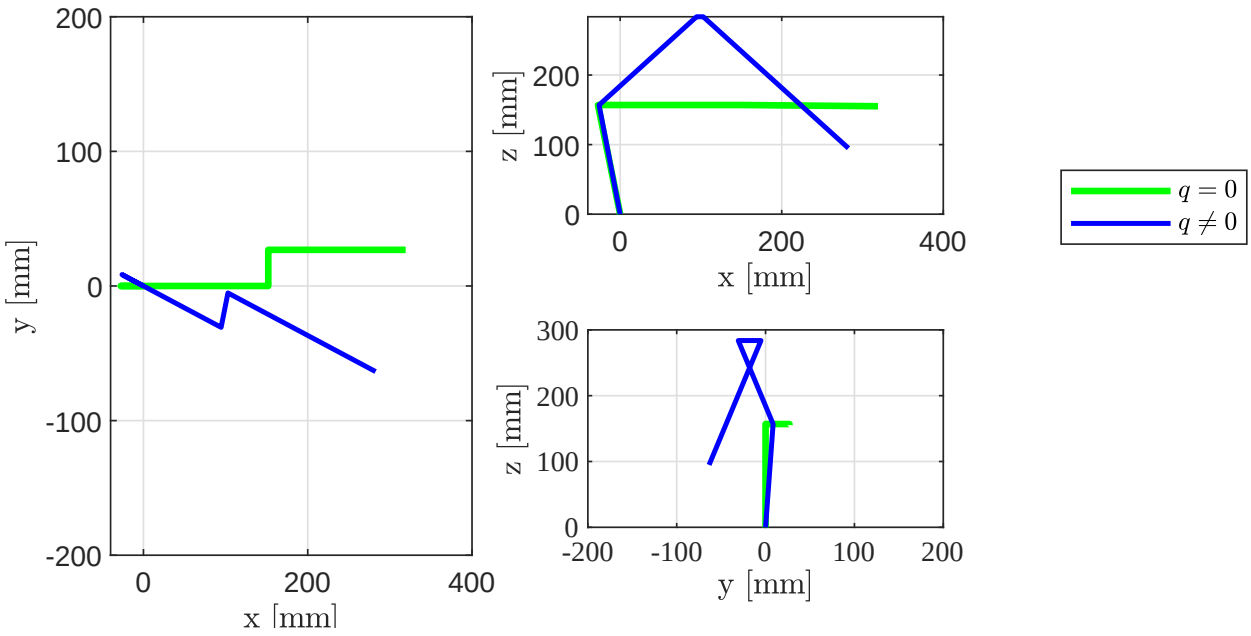

Figure 2.7: Transformation Matrix Validation

With the kinematic diagram in Figure 2.6 and the given inputs from Table 2.3 in mind, the robot arm visualization in Figure 2.7 looks correct, and the transformation matrix can be used.

2.3 Forward Kinematics Using Screw Theory

There are several ways to describe the kinematics of a robot manipulator, this section will present the forward kinematics using screw theory. The fixed frame is presented as $\{s\}$ and the end effector frame is $\{b\}$. Each joint have their own screw axis S_i.

From the DH - Representation in the previous section, it is presented the transformation matrix $\mathbf{T}_0^4$, including the translation $\mathbf{Tr}_0^4$ between the two coordinate system, but also the rotation $\mathbf{R}_0^4$. Now, we are representing the robotic manipulator using screw theory, but similarly the transformation and rotation between the two coordinate system are used.

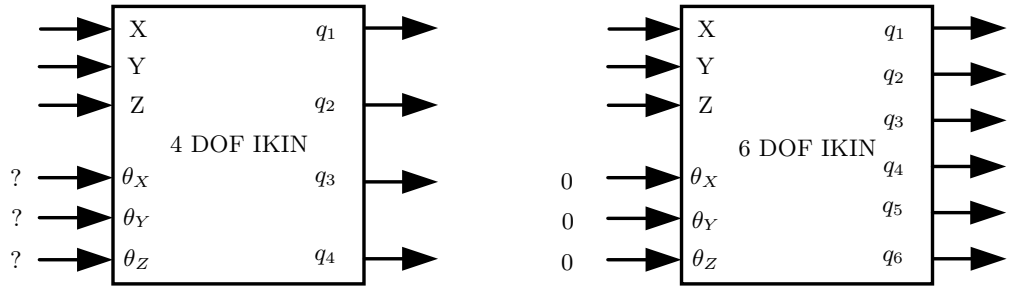

Figure 2.8: Inverse Kinematics 4 vs 6 DOF

Now, imagine we are using the inverse kinematics to find the joint configuration, we would then have to know the rotation of the coordinate system at the end effector with respect to the base. Looking at the left side of Figure 2.8 and Figure 2.9, representing the 4 DOF manipulator, it is

difficult to find rotations matching the given configuration, when the frame coordinate is fixed in parallel with the prismatic joint.

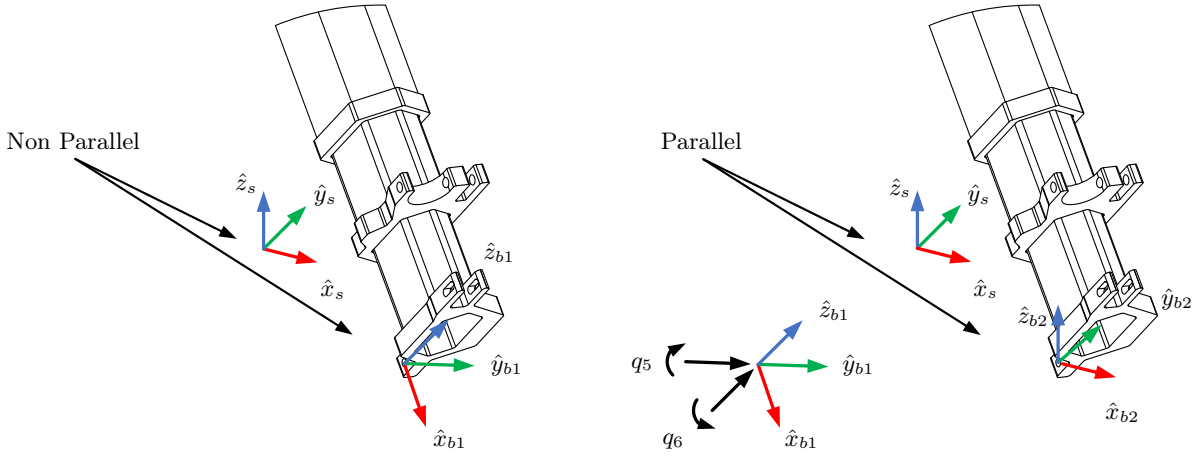

Figure 2.9: End effector 4 vs 6 DOF

Lets now imagine there was a way to fix the orientation of the end effector frame with respect to the ground. Then the input of the rotation could be constant, for example 0, stating that end frame is parallel to ground frame at all time. By adding two revolute joints in the end effector, this is accomplished, and the inverse kinematics can be solved. The right side of Figure 2.8 and Figure 2.9 is illustrating this. The two extra joints are rotating around $\hat{y}_{b1}$ and $\hat{z}_{b1}$. There is no need to rotate around $\hat{x}_{b1}$, because the telescope q_4 only translates, not rotate.

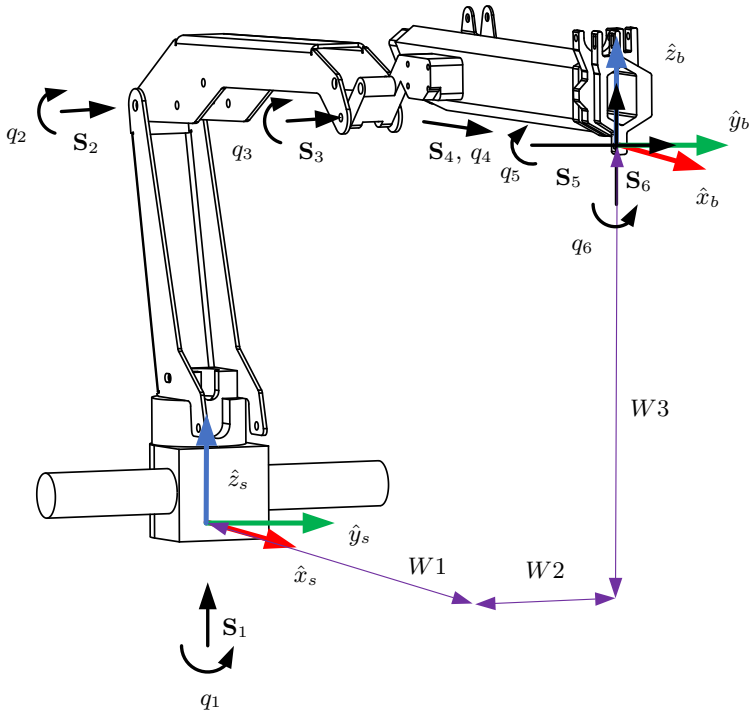

Figure 2.10: Robotic Manipulator in Rest Position

Figure 2.10 shows all the joints and screw axes, including the two extra joints $q_5 - q_6$. The axis of rotation for these two axes is chosen intuitively, to be able to compensate for the other joints. Looking at the Figure, it is clear that $q_6 = -q_1$ and $q_5 = -(q_2 + q_3)$ at for all joint configurations, as long as frame {b} is parallel to {s}.

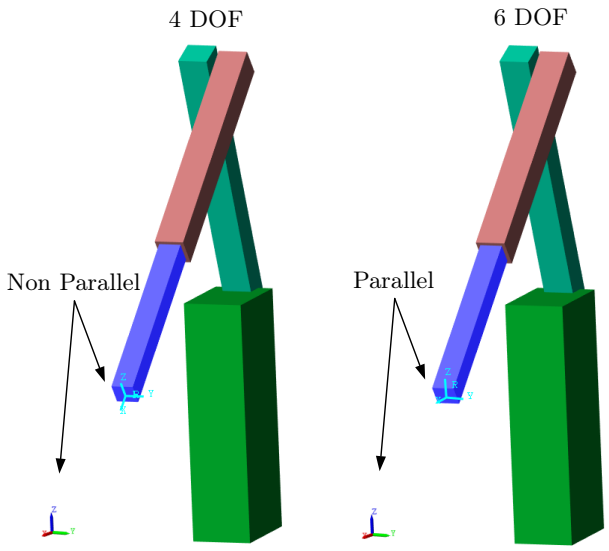

Figure 2.11: Simulink 4 vs 6 DOF

The two extra revolute joints are added to the 4 DOF configuration. From Simulink tests it is clear that the end effector coordinate is parallel for the 6 DOF at all time. The results is visualized in Figure 2.11. The Simulink model build up, and inverse kinematics implementation is explained later in this chapter.

As an alternative to the transformation and rotation defined by the transformation matrix using the DH representation, let's introduce the matrix $\mathbf{M}$ representing the rotation and the translation from crane frame $\{s\}$, to end effector frame $\{b\}$ when the crane is in initial position, meaning q_1 - $q_4 = 0$. Since the two frames are parallel, there is no rotation. However, $W1$, $W2$, $W3$ is the translation in X,Y,Z respectively. From Figure 2.10 matrix $\mathbf{M}$ can be written as:

$$\mathbf{M} = \begin{bmatrix} 1 & 0 & 0 & W1 \\ 0 & 1 & 0 & W2 \\ 0 & 0 & 1 & W3 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{2.11}$$

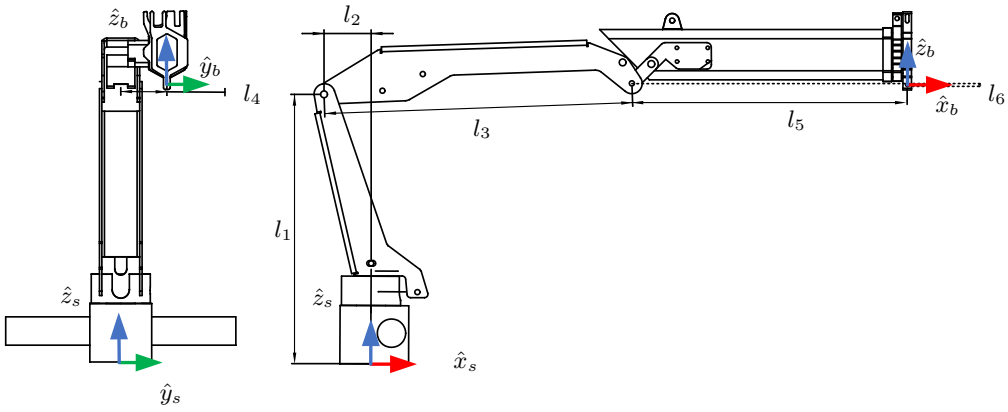

Figure 2.12: Distance Between Joints

From Figure 2.12, $W1$, $W2$, and $W3$ are calculated as:

$$W1 = -l_2 + l_3 + l_5$$
$$W2 = l_4$$
$$W3 = l_1 - l_6$$

Then the screw axis for the revolute joints is defined as:

$$\mathbf{S}_i = \begin{bmatrix} \mathbf{sw}_i \\ \mathbf{sv}_i \end{bmatrix} \tag{2.12}$$

$\mathbf{sw}_i$ is the unit vector parallel to the screw vector (for all the revolute joints):

$$\mathbf{sw}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{sw}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{sw}_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{sw}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \quad \mathbf{sw}_6 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{2.13}$$

Next step is then to describe the point of each screw axis, with respect to the crane frame $\{s\}$. The positions are presented as vectors $\mathbf{a}_1$ - $\mathbf{a}_6$ (for all the revolute joints):

$$\mathbf{a}_1 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{a}_2 = \begin{bmatrix} -l_2 \\ 0 \\ l_1 \end{bmatrix}, \quad \mathbf{a}_3 = \begin{bmatrix} -l_2 + l_3 \\ 0 \\ l_1 \end{bmatrix}, \quad \mathbf{a}_5 = \begin{bmatrix} W1 \\ W2 \\ W3 \end{bmatrix}, \quad \mathbf{a}_6 = \begin{bmatrix} W1 \\ W2 \\ W3 \end{bmatrix} \tag{2.14}$$

$\mathbf{sv}_i$ is found by crossing the position vectors with the direction vectors: $\mathrm{sv}_i = \mathbf{a}_i \times \mathbf{sw}_i$

$$\mathbf{sv}_1 = \mathbf{a}_1 \times \mathbf{sw}_1, \quad \mathbf{sv}_2 = \mathbf{a}_2 \times \mathbf{sw}_2, \quad \mathbf{sv}_3 = \mathbf{a}_3 \times \mathbf{sw}_3, \quad \mathbf{sv}_5 = \mathbf{a}_5 \times \mathbf{sw}_5, \quad \mathbf{sv}_6 = \mathbf{a}_6 \times \mathbf{sw}_6 \tag{2.15}$$

Then the $\mathbf{S}_i$ vectors can be constructed for all the revolute joints, by inserting the vectors from Equation 2.13 and 2.15 in to Equation 2.12. However, for the translation joint, there is no rotation, but translation along the axis $\hat{x}_s$ (again with all other joints = 0). Therefore, the screw axis is given by:

$$\mathbf{S}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \tag{2.16}$$

All the screw axes can be presented:

$$\mathbf{S}_1 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{S}_2 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -l_1 \\ 0 \\ -l_2 \end{bmatrix} \quad \mathbf{S}_3 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -l_1 \\ 0 \\ l_3 - l_2 \end{bmatrix} \quad \mathbf{S}_4 = \begin{bmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{S}_5 = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -W3 \\ 0 \\ W1 \end{bmatrix} \quad \mathbf{S}_6 = \begin{bmatrix} 0 \\ 0 \\ 1 \\ W2 \\ -W1 \\ 0 \end{bmatrix} \tag{2.17}$$

Now, using the product of exponential, the tool-point position and orientation can be calculated based on all the angles q_1 - q_6. Note that q_5 and q_6 are dummy joints and only affects the orientation of the tool point frame {b}, not the position. The forward kinematics can then be expressed as a product of exponential (POE) as:

$$\mathbf{T(q)} = \mathbf{T} = e^{[\mathbf{S}_1]q_1} e^{[\mathbf{S}_2]q_2} ... e^{[\mathbf{S}_6]q_6} \mathbf{M} \tag{2.18}$$

Calculate the matrix $\mathbf{T(q)}$ using the same angles as with the DH-parameter, the method can be validated. To calculate $\mathbf{T}$ the provided Matlab function T = FKinSpace(M,Slist,thetalist) is used. The tool-point position {b} in Cartesian space with respect to the robot base frame {s} is giving the same result as with the DH method, validating this method as well for forward kinematics. Observe that the the transformation in Equation 2.18 requires a value from the 6 joints, and is always giving one solution.

The forward kinematics can also be written as:

$$\mathbf{x} = f(\mathbf{q}), \quad \mathbf{x} = \begin{bmatrix} X \\ Y \\ Z \\ \theta_X \\ \theta_Y \\ \theta_Z \end{bmatrix}, \quad \mathbf{q} = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \\ q_4 \\ q_5 \\ q_6 \end{bmatrix} \tag{2.19}$$

2.4 Velocity Kinematics Using Space Jacobian

The velocity kinematics represent the relationship between the end effector velocity in Cartesian space, and the joint velocity in joint space. In robotics, the relationship between these two is presented with the help of a the Jacobian matrix $\mathbf{J}$. If the position and orientation in Cartesian space is represented as $\mathbf{x}$, and the joint positions are represented as $\mathbf{q}$, their relationship can be presented as:

$$\dot{\mathbf{x}} = \mathbf{J}\dot{\mathbf{q}} \tag{2.20}$$

For the purpose of this project we are interested in the space Jacobian. The space Jacobian $\mathbf{J}_s$ connects the joint velocities $\dot{\mathbf{q}}$ to the spatial twist $\mathbf{V}_s$. The spatial twist $\mathbf{V}_s$ is a vector including the Cartesian velocity expressed in the crane base frame {s}. $\mathbf{V}_s$ is a 6x1 vector including the angular velocity $\boldsymbol{\omega}$ and linear velocity $\boldsymbol{v}$. Again, the linear velocity expressed in the {s} frame, $\boldsymbol{v}_s$ is the part interesting for this crane.

$$\mathbf{V}_s = \begin{bmatrix} \boldsymbol{\omega}_s \\ \boldsymbol{v}_s \end{bmatrix}_{6x1} = \mathbf{J}_s\dot{\mathbf{q}} \tag{2.21}$$

The space Jacobian can be derived from the POE formula in Equation 2.18. The spatial twist is $\mathbf{V}_s = \dot{\mathbf{T}}\mathbf{T}^{-1}$. First lets differentiate $\mathbf{T}$.

$$\dot{\mathbf{T}} = [\mathbf{S}_1]\dot{q}_1 e^{[\mathbf{S}_1]q_1} ... e^{[\mathbf{S}_6]q_6}\mathbf{M} + e^{[\mathbf{S}_1]q_1}[\mathbf{S}_2]\dot{q}_2 e^{[\mathbf{S}_2]q_2} ... e^{[\mathbf{S}_6]q_6}\mathbf{M} + ... \tag{2.22}$$

Taking the inverse of matrix $\mathbf{T}$ gives:

$$\mathbf{T}^{-1} = \mathbf{M}^{-1} e^{-[\mathbf{S}_6]q_6} ... e^{-[\mathbf{S}_1]q_1} \tag{2.23}$$

Setting together Equation 2.22 and 2.23, and write in vector form gives:

$$\mathbf{V}_s = \dot{\mathbf{T}}\mathbf{T}^{-1} = \mathbf{J}_{s1} + \mathbf{J}_{s2}(\mathbf{q})\dot{q_1} + ... + \mathbf{J}_{s6}(\mathbf{q})\dot{q_6} \tag{2.24}$$

Write in matrix form:

$$\mathbf{V}_s = [\mathbf{J}_{s1} \quad \mathbf{J}_{s2}(\mathbf{q}) \quad ... \quad \mathbf{J}_{s6}(\mathbf{q})] \begin{bmatrix} \dot{q_1} \\ \vdots \\ \dot{q_6} \end{bmatrix} = \mathbf{J}_s(\mathbf{q})\dot{\mathbf{q}} \tag{2.25}$$

Later in this report, when controlling the crane actuators, the linear Cartesian velocity for the end-effector is given by us and used to calculate the joint velocities. Then we are using the relationship:

$$\dot{\mathbf{q}} = \mathbf{J}_s(\mathbf{q})^{-1}\mathbf{V}_s \tag{2.26}$$

Because the Jacobian matrix is depending on the joint positions $\mathbf{q}$ we are depending on the joint position from the sensors to determine the joint velocities.

2.5 Inverse Position Kinematics Using Screw Theory

As discussed previously the forward kinematics is calculating the end effector position in Cartesian space, based on the value of each joint. This can be done as discussed using the POE Equation 2.18 from the forward kinematics section. Now, the calculations are going the other way. We want to find a value for each joint to reach a certain point i Cartesian space.

For the case of the POE equation including the two dummy joints q_5 and q_6, we have 6 equation with 6 unknowns. However, for the control of the crane in a later stage, we do not care about the orientation of the end effector, only the position. Meaning that in reality we have 3 desired positions in Cartesian space and 4 unknown joint positions to control. In other words, multiple solutions are satisfying our desired end effector position. Figure 2.13 is illustrating this challenge showing two different configurations with the same end effector position. One of the configuration have the prismatic joint fully retracted, and the other have some extension on the prismatic joint, but the other joint compensate with a larger angle to reach the same point in Cartesian space. This is called actuator redundancy.

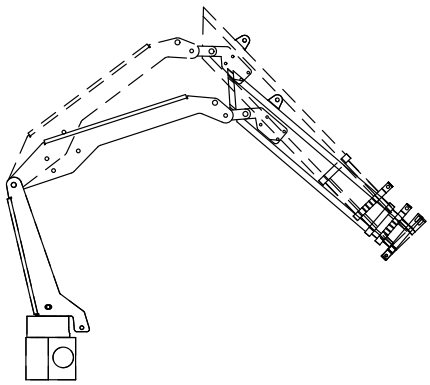

Figure 2.13: Actuator Redundancy

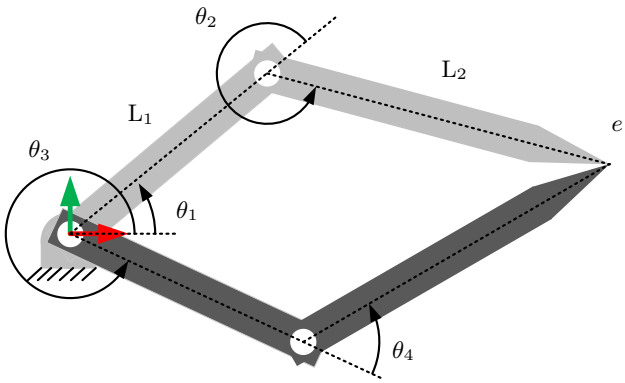

Figure 2.14: Elbow Up Elbow Down

Figure 2.14 illustrates a simple two dimential revolute joint robotic manipulator to illustrate two different joint configurations with the same end effector position. This is a different case than for the crane, because here there are two equations with two unknowns, but the trigonometry has two solutions.

To solve the inverse kinematics analytical is in some cases possible and can also be simple for robotic manipulators with few bodies like the one presented in Figure 2.14. In a robot control perspective, if the inverse kinematics was solved analytically, some algorithm to choose the right solution is necessary. However, a more common method is to solve the inverse kinematics numerically. This is done with some root finding method, and initial guess close to the desired solution. This is often done with a method like Newton-Raphson method, where a initial guess close to the solution is used.

Lets introduce the function $g(q)$ and assume we want to solve $g(q) = 0$. Using the Newton-Raphson method, lets write the Taylor expansion first:

$$g(q) = g(q^0) + \frac{\partial g}{\partial q}(q^0)(q - q^0) + \text{higher order terms} \tag{2.27}$$

Removing the higher terms, and setting $g(q) = 0$:

$$q = q^0 - \left(\frac{\partial g}{\partial q}(q^0)\right)^{-1} g(q^0) \tag{2.28}$$

If the process is repeated the iterative solution can be written as:

$$q^{i+1} = q^i - \left(\frac{\partial g}{\partial q}(q^i)\right)^{-1} g(q^i) \tag{2.29}$$

Now the iterations can go on until the error $\epsilon = |q^i - q^{i+1}|$ is smaller than a threshold of our choice.

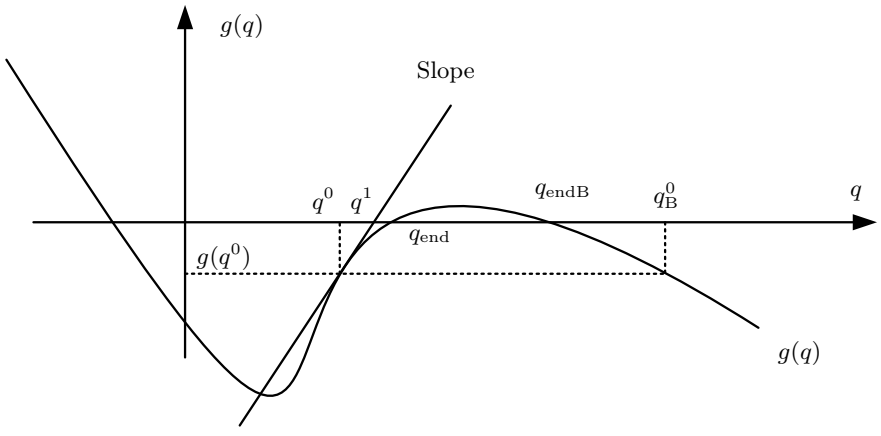

Figure 2.15: Newton-Raphson Illustration

Figure 2.15 illustrates the Newton Raphson method with first iteration on the function $g(0) = 0$. First iteration q^0 use the slope of $g(q_0)$ to reach the second iteration q_1. Repeating this process and the solution is converging to q_{end}. Lets now imagine instead that the initial guess is q^0_{B}. From the Figure we can assume this solution is converging to q_{endB}. This is a different solution and this

example stresses the importance of the initial guess in the numerical root finding Newton-Raphson method.

In our case there is not only one equation we want to find the root, but a set of equation. The same formula works, also when the set of equation is set into matrix form. Now the iterative process to be able program the numerical inverse is presented. First, lets recall that our end effector coordinates are a function of our joint position, $\mathbf{x} = f(\mathbf{q})$. The goal is to solve $g(\mathbf{q}) = \mathbf{x}_d - f(\mathbf{q}_d) = 0$, and our interest is the joint positions $\mathbf{q}_d$. Assume the initial guess close to the solution, the Taylor expansion can be written as:

$$\mathbf{x}_d = f(\mathbf{q}) = f(\mathbf{q}^0) + \underbrace{\frac{\partial g}{\partial \mathbf{q}}|_{\mathbf{q}^0}}_{J(\mathbf{q}^0)} \underbrace{(q_d - q^0)}_{\Delta \mathbf{q}} + \text{higher order term} \tag{2.30}$$

$\mathbf{J(q^0)}$ is the Jacobian evaluated at $\mathbf{q}^0$. Removing the higher order terms, the equation becomes:

$$\mathbf{J}(\mathbf{q}^0)\Delta\mathbf{q} = \mathbf{x}_d - f(\mathbf{q}^0) \tag{2.31}$$

If J($\mathbf{q}^0$) is of size $m \times m$:

$$\Delta\mathbf{q} = \mathbf{J}^{-1}(\mathbf{q}^0)(\mathbf{x}_d - f(\mathbf{q}^0)) \tag{2.32}$$

Because the Jacobian matrix is not always square, often the Moore-Penrose pseudoinverse is used to find an approximate solution to the inverse. The pseudoinverse also have an advantage of handling singularities. In other word, when the task is to solve different numerical inverse Jacobian matrices in a moving robotic manipulator, it is an advantage that there is no complex numbers as output. The Moore-Penrose pseudoinverse can be computed with the pinv function [19] in Matlab.

The twist $\mathbf{V}_s$ can be used as a quantification between the desired position and the initial guess. Therefore, by substituting Equation 2.32.

$$\Delta\mathbf{q} = J^{\dagger}\mathbf{V}_s \tag{2.33}$$

When calculating the pseudoinverse it is an advantage to keep in mind, how many variables, and how many equations. For example for this crane, there are 6 joints and 6 equation, meaning $\mathbf{J}(\mathbf{q})_{6\text{x}6}$, but the rank of the matrix is 5. This means Jacobian is "fat". When the Jacobian is "fat", the pseudoinverse finds a solution among the infinite possible solutions by minimizing the least squares error [19]. In other words, the pseudoinverse finds the solution that minimizes the sum of the squares of the differences between the desired and and initial guess for the joints q_1 - q_6.

2.6 6 and 5 DOF to 4 and 3 DOF

From the kinematics explained above, the adding of extra joint was introduced. Also actuator redundancy with more actuators than coordinates was mentioned. Looking at the left of Figure 2.16, there are 6 input variables X,Y,Z, θ_x,θ_y, θ_z and 6 outputs $q_1 - q_6$. However, from the system $\theta_x = 0$ is automatically obtained, because no joints can rotate around that axis, and is the reason the rank of the Jacobian is 5 and not 6. This presents the actuator redundancy, by having 5 manipulative inputs and 6 outputs. By removing one output, the prismatic joint q_4, there are 5 manipulative inputs and 5 outputs, as presented in the right side of Figure 2.16.

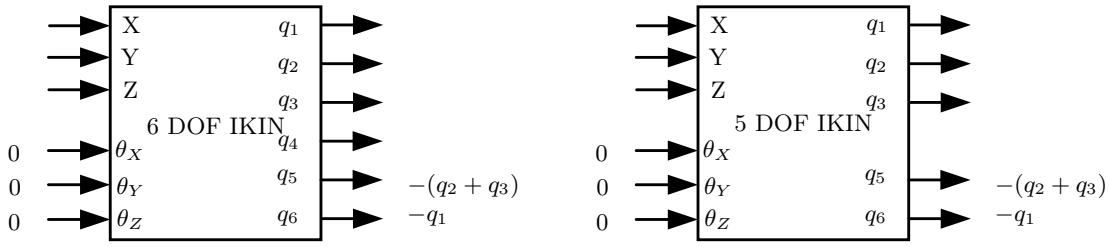

Figure 2.16: 6 DOF to 5 DOF with locked telescope

Knowing the coordinate systems {b} and {s} are parallel, the output of the inverse kinematics for q_5 and q_6 will always be constrained by q_1, q_2 and q_3. Therefore, the 6 DOF problem is having 3 manipulative coordinates in X, Y, and Z and 4 DOF $q_1 - q_4$. This is illustrated in the left side of Figure 2.17 as well.

Now removing q_4, meaning the telescope was locked in a constant position. Then, the manipulator could have different configuration in the form of elbow up and elbow down, but not actuator redundancy as discussed with the active telescope. This is illustrated on the right side in Figure 2.17 and can be referred to as the 3 DOF, and is matching the 5 DOF explanation above.

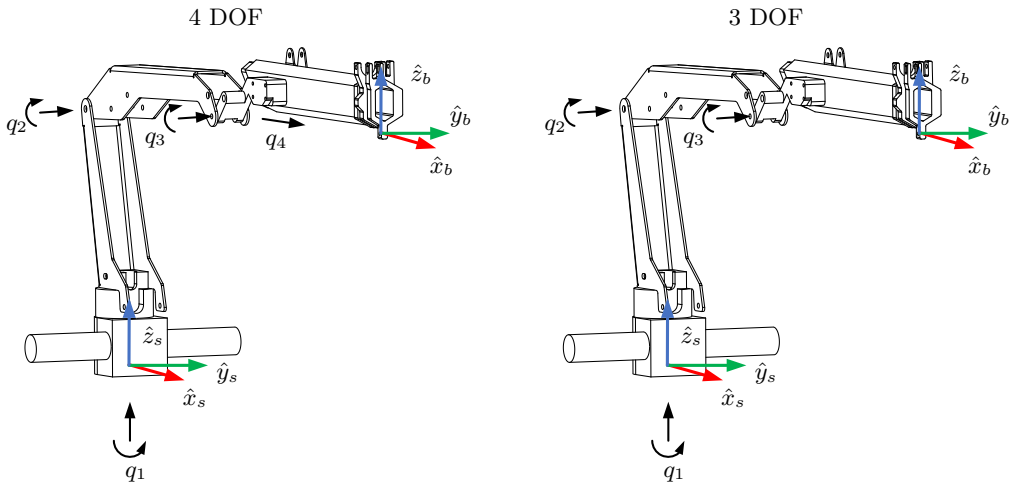

Figure 2.17: 4 DOF vs 3 DOF

The dummy joints q_5 and q_6 are still there in the inverse kinematics, to be able to solve them numerically, but they are not used to anything in the control system later. Therefore we can use 4 and 3 DOF from here in the report.

2.7 Knuckle Joint Angle Relations

In a later stage of this report, the sensors are be presented. Due to more convenient mounting of the sensor, the angle q_3, is measured on revolute joint I instead of joint K. To have the exact angle of K, the relationship between them needs to be found. For the trigonometric relationships in a knuckle joint, the calculations done are highly inspired from a similar joint presented in the paper:"Development of 3D Anti-Swing Control for Hydraulic Knuckle Boom Crane" [17]. The knuckle joint is presented in Figure 2.18.

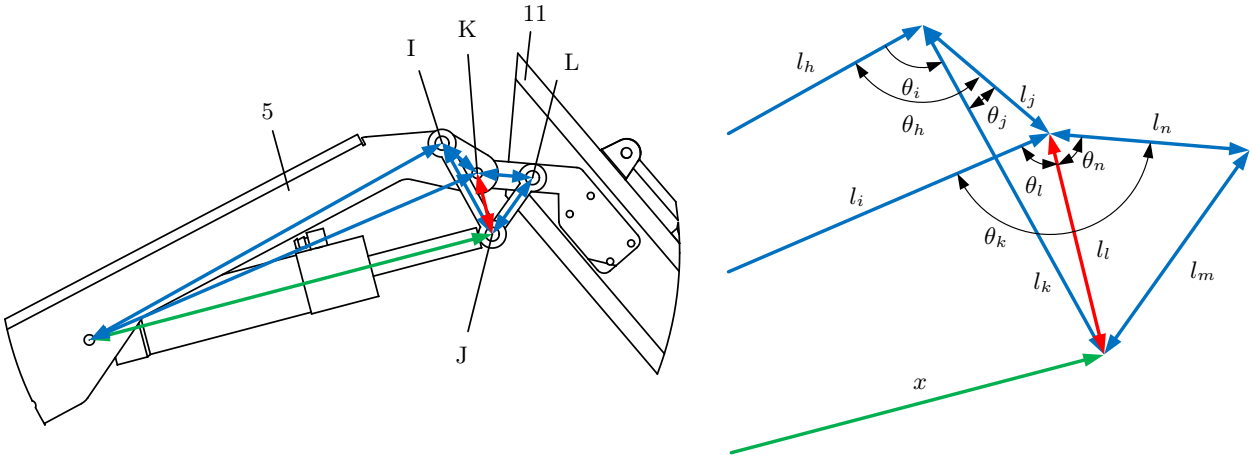

Figure 2.18: Knuckle Joint

The desired output from this calculation is to find angle θ_k based on the angle θ_i. θ_k is directly proportional with q_3.

$$\theta_h = \cos^{-1}\left(\frac{l_h^2 + l_j^2 - l_i^2}{2l_h l_i}\right) \tag{2.34}$$

Then the distance x based on the angle θ_i:

$$x = \sqrt{l_h^2 + l_k^2 - 2l_h l_k \cos(\theta_i)} \tag{2.35}$$

$$\theta_j = \theta_h - \theta_i \tag{2.36}$$

$$l_l = \sqrt{l_j^2 + l_k^2 - 2l_j l_k \cos(\theta_j)} \tag{2.37}$$

$$\theta_n = \cos^{-1}\left(\frac{l_l^2 + l_n^2 - l_m^2}{2l_l l_n}\right) \tag{2.38}$$

$$\theta_l = \cos^{-1}\left(\frac{l_i^2 + l_l^2 - x^2}{2l_i l_l}\right) \tag{2.39}$$

$$\theta_k = \theta_n + \theta_l \tag{2.40}$$

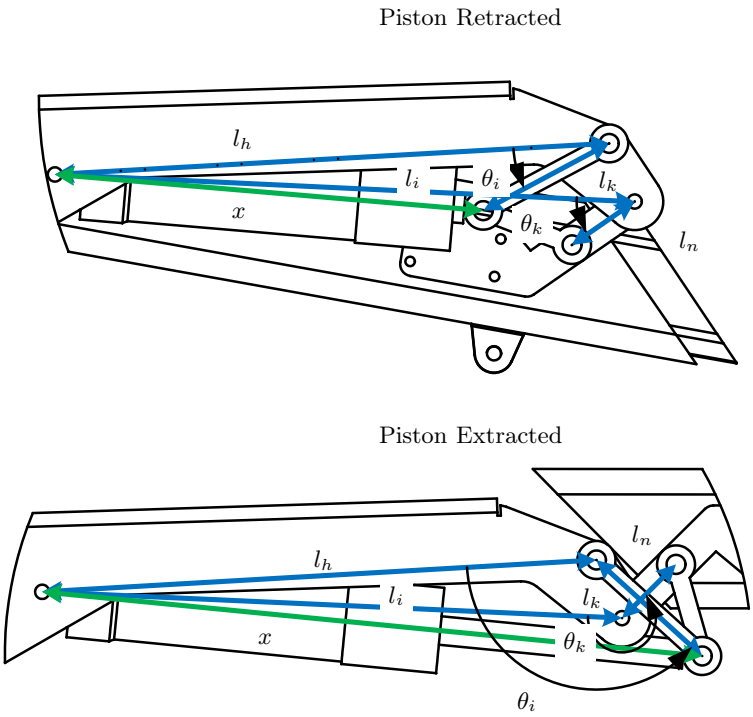

Figure 2.19: Knuckle Joint Angles

Figure	**x**	θ_i	θ_k
Fig 2.19 Top	90.3 [mm]	24.9°	37.4°
Fig 2.19 Bottom	139.3 [mm]	134.8°	227.5°

Table 2.4: Measured angles in Solidworks

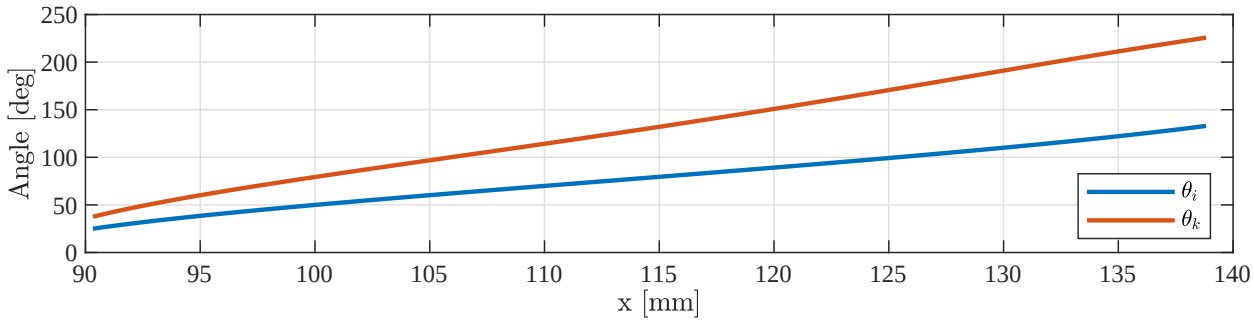

Figure 2.20: Knuckle Joint Angle Relations Calculated

With the equations above, a function is created in Matlab B.3. This function takes the input from θ_i and outputs θ_k and piston length x. By looping through the function from 25° to 133°, the piston length x, and the two angles are plotted in Figure 2.20. Also, the angles are measured from the Solidworks Assembly with the two configuration presented in Figure 2.19 and presented in Table 2.4. The angles measured, and the angles calculated, are equal within 0.1 [deg] for these two configurations. This means that the mathematical model of the knuckle joint angles is valid.

$$q_3 = \theta_k - \theta_{k,init}, \quad \theta_{k,init} = \frac{227.5}{180}\pi = 3.9706[rad] \tag{2.41}$$

The function used to implement Equation 2.41 to the IPC for is added in appendix B.4.

2.8 Frame Coordinate and Crane Coordinate

To be able to heave-compensate when the crane is exposed to disturbance, the relation between the fixed frame and the crane frame needs to be established. The coordinate frame {s} used in the kinematics explained earlier was corresponding to what we then called a fixed frame. However, when we introduce displacement to the body where the crane is mounted, we need to define a new fixed coordinate frame. This frame is located in the revolute joint between the body connected to the crane, and the frame itself. We call this the {f} coordinate frame. Later when the crane end-effector **e** is controlled with respect to the Cartesian coordinate frame, the frame {f} is the reference. This is so the end effector **e** have the same distance to the ground, and stay in the same position with respect to the frame, even if the handle is used to "disturb" the crane itself.

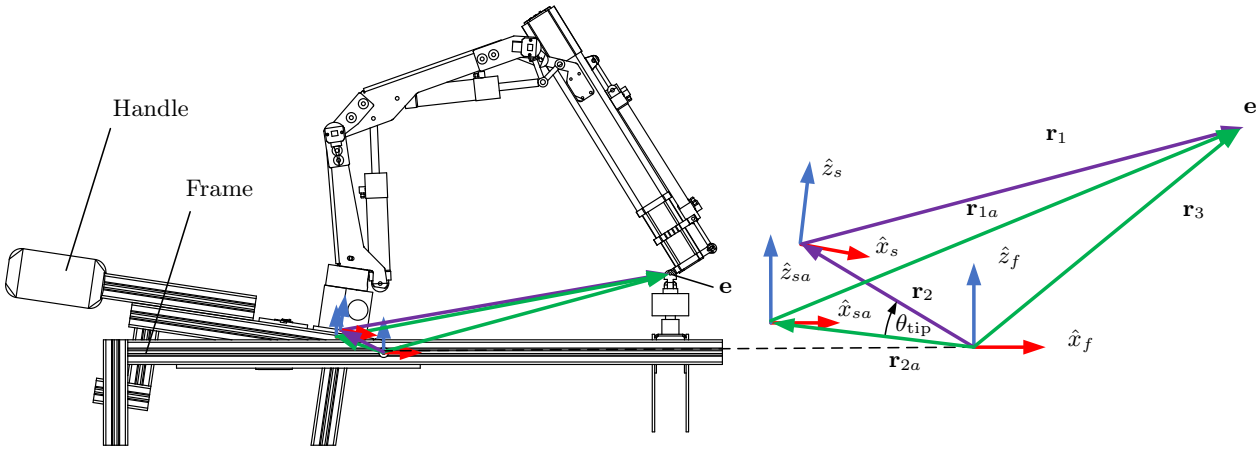

Figure 2.21: Crane Coordinate to Frame Coordinate

The crane can only be rotated around the y coordinate $\hat{y}_f$ but there is translation in $\hat{x}_f$ and $\hat{z}_f$ as a result of the rotation. Figure 2.21 illustrates the relation between the two coordinate systems. The green vectors are representing the connection between the two coordinate systems and the end effector when $\theta_{\text{tip}} = 0$, and the purple vectors are representing the relation when $\theta_{\text{tip}} \neq 0$. The illustration shows only two dimensions even if the y axis is also included in the r vector. This is because the y value is always the same value for both coordinate system and does not need any translation.

Lets imagine the crane is not moving, but we want to keep the the end effector at the same point in frame coordinate, after a potential disturbance. Therefore $\mathbf{r}_3$ needs to be calculated before the tilt θ_{tip}, because this is the only vector that is the same before ($\theta_{\text{tip}} = 0$) and after the tilt ($\theta_{\text{tip}} \neq 0$), and can be used to calculate vector r_1. From figure 2.21 it is clear that:

$$\mathbf{r}_3 = \mathbf{r}_{2a} + \mathbf{r}_{1a} \tag{2.42}$$

$\mathbf{r}_{2a}$ is a constant vector in [mm]:

$$\mathbf{r}_{2a} = [-37,\ 0,\ 13] \tag{2.43}$$

Rotation around y can be expressed as:

$$\mathbf{R}_y(\theta_{\text{tip}}) = \begin{bmatrix} \cos\theta_{\text{tip}} & 0 & \sin\theta_{\text{tip}} \\ 0 & 1 & 0 \\ -\sin\theta_{\text{tip}} & 0 & \cos\theta_{\text{tip}} \end{bmatrix} \tag{2.44}$$

Based on the angle $\theta_{\text{tip}} \neq 0$ lets find the vector $\mathbf{r}_2$ expressed in frame coordinates {f}:

$$\mathbf{r}_2 = \mathbf{R}_y(\theta_{\text{tip}})\mathbf{r}_{2a} \tag{2.45}$$

Based on the angle $\theta_{\text{tip}} \neq 0$ lets find the vector $\mathbf{r}_1$ expressed in frame coordinates {f}:

$$\mathbf{r}_1 = \mathbf{r}_3 - \mathbf{r}_{2b} \tag{2.46}$$

Now, to find the desired end effector location in crane coordinates {s} it is important to rotate in negative θ_{tip} direction:

$$\mathbf{r}'_1 = \mathbf{R}_y(-\theta_{\text{tip}})\mathbf{r}_1 \tag{2.47}$$

To control the crane back to the desired end effector position in {f} the desired position in {s}, $\mathbf{r}'_1$ is used as input in the inverse kinematics calculation.

2.9 Kinematics Control Using Matlab Simulink

2.9.1 Simulink Model

To be able to simulate the kinematics before the hardware implementation, Matlab Simulink with Simscape Multibody toolbox is used. Before the kinematic equation can be tested, the bodies and joint representing the crane needs to be constructed in the Simulink environment. Just like in the kinematic calculations, the simplified crane representation includes 3 revolute joints and 1 joint. Blocks for the first part are shown in Figure 2.22. Simscape multibody works in such a way, that the coordinate frames are rotated and translated from ground to the end of the crane, and joints and bodies are added to the right frames.

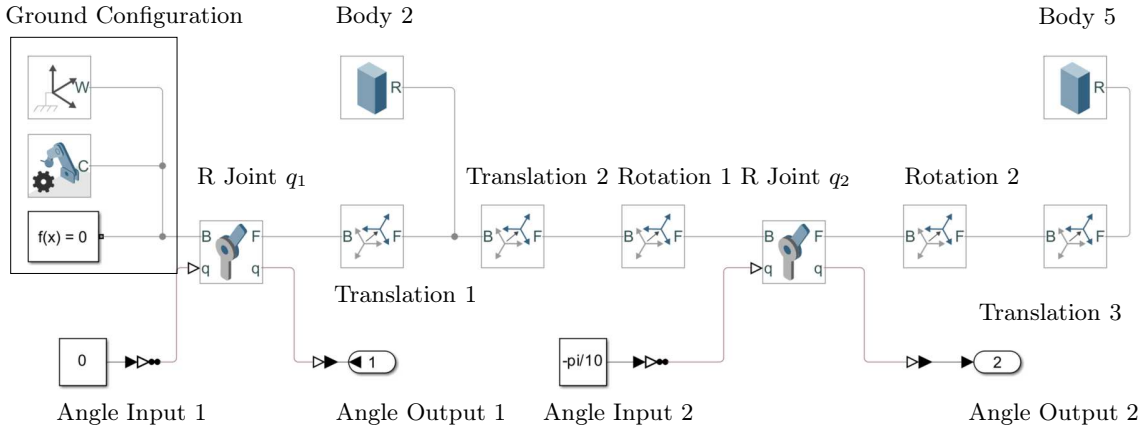

Figure 2.22: Multibody Dynamics Simulink

For all multibody models in Simulink, the 3 blocks in Ground Configuration is needed as a reference point. The blocks are named "World Frame", "Mechanism Configuration" and "Solver Configuration" respectively. Revolute joints are represented in block "Revolute Joints", and is always rotating around the z axis. prismatic joint are represented with "Prismatic Joint" block and is translating along the z axis. Transformations and rotations are represented by "Rigid Transform" blocks. A number of rigid transformations are necessary to ensure that all the bodies and joints are oriented and positioned correctly.

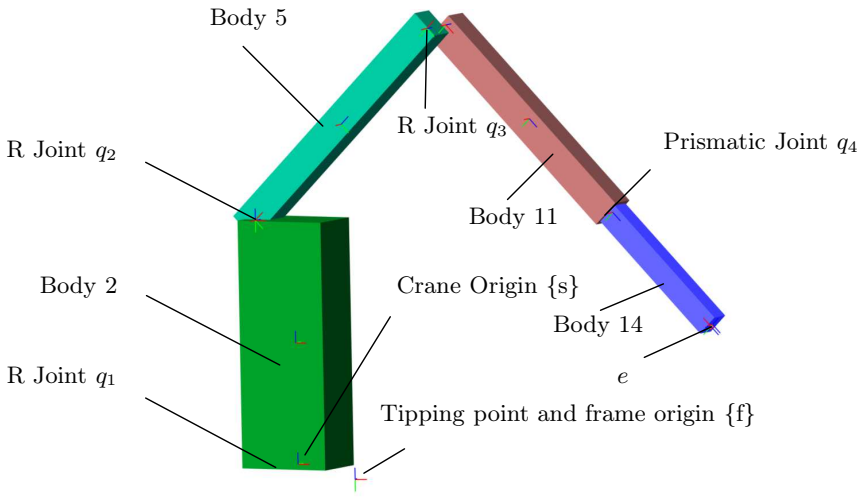

Figure 2.23: Complete Visualization in Simulink

Variable	Value
q_1	$-\pi/10$ [rad]
q_2	$-\pi/4$ [rad]
q_3	$\frac{\pi}{2}$ [rad]
q_4	100 [mm]

Figure 2.24: Input Variables Simulink

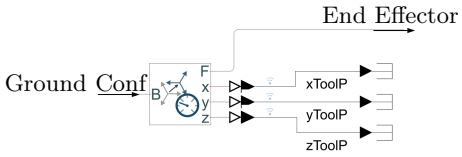

Figure 2.25: Toolpoint Sensor

Figure 2.23 is illustrating the resultant manipulator visualization. The angles of each joint input is presented in the table in Figure 2.24. Note that q_4 is divided by 1000, to convert from [mm] to [m] before actuating the prismatic joint block. If we compare with the robotic manipulator presented at initial condition, both in the DH parameter chapter and the forward kinematics using screw theory, all joints are presented as expected, meaning the mechanical model space representation in simulation is ready for inverse kinematics testing in the simulation. Note that both dummy joints and the crane frame {f} is added in Figure 2.23. To validate the inverse kinematics it collect data from the end effector position. This is done with the transform sensor block presented in Figure 2.25.

2.9.2 Matlab Functions in Simulink

Matlab functions can be used in the Simulink environment using function blocks. This is convenient for testing the functions, and the kinematics. Later, when the functions, kinematics and control is implemented in the IPC controller, Matlab can directly export the function to PLC code, readable for the IPC. Figure 2.26 illustrates the two function blocks Getr3 and Getr1b used to calculate the new $\mathbf{r}_1$.

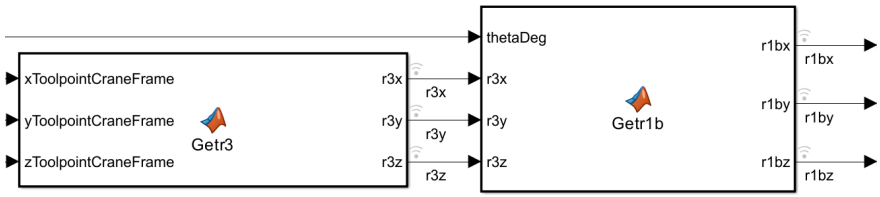

Figure 2.26: Function Block in Simulink

2.9.3 Forward Kinematics And Model Comparation Simulink

The function block for forward kinematics presented in Figure 2.27, includes the code presented in appendix B.5. This is a copy of the function given in "Modern Robotics" book as FKinSpace(), adjusted to fit this robotic manipulator. A ramp function is given as input to this function to be able to plot the result of several different joint inputs. The same input signal is sent to the mechanical model in Simulink, using the block shown in the Figure as A, B, C, and D.

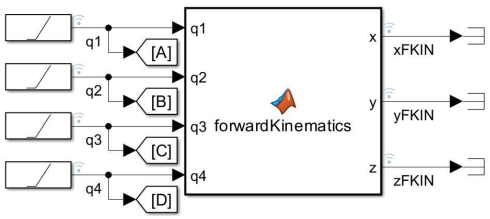

Figure 2.27: Forward Kinematic Block Simulink

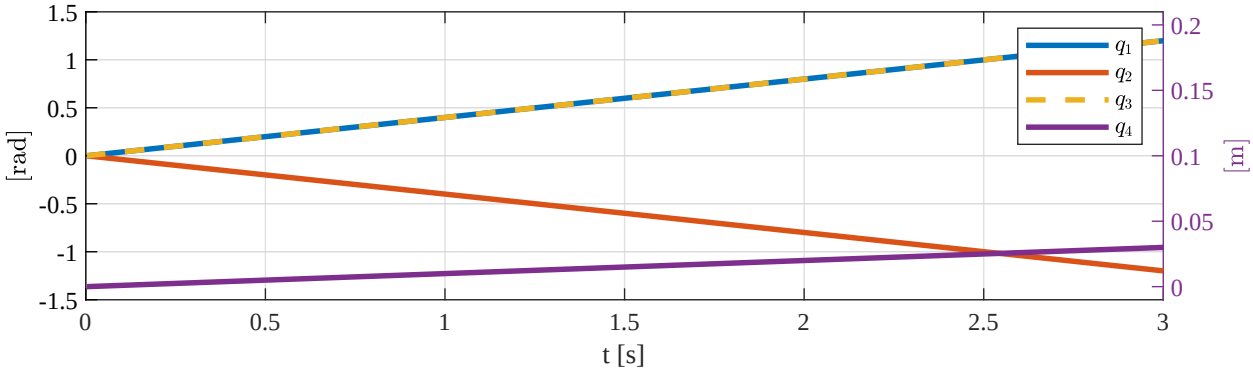

Figure 2.28: Forward Kinematics Joint Positions

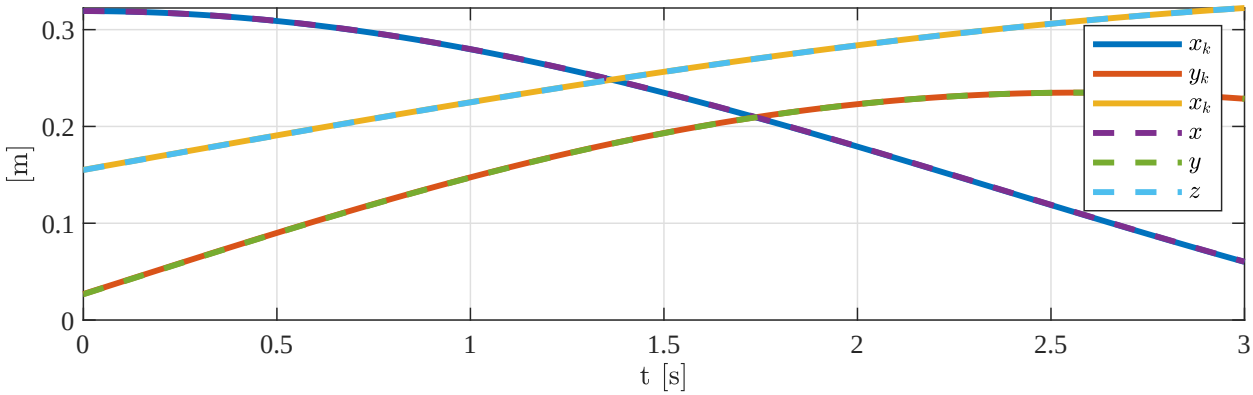

Figure 2.29: Forward Kinematics Cartesian {s} End Effector Position

The end effector position presented in the Cartesian coordinate {s} is measured with the sensor on the crane simulation, and logged from the output of the forward kinematics function block. The result is plotted in Figure 2.29 where x_k, y_k, and z_k is the kinematic calculated solution, and x, y, and z is the measured position in the simulation environment. The visualization for the presented motion is shown in 2.30.

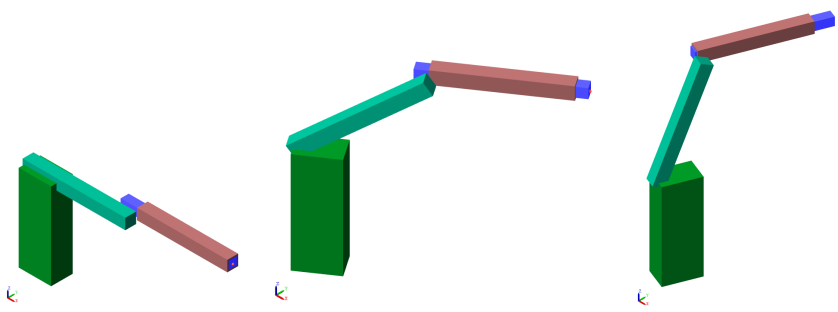

Figure 2.30: Forward Kinematics Crane Visualization

2.9.4 Simulate Inverse Position Kinematics with Actuator Redundancy

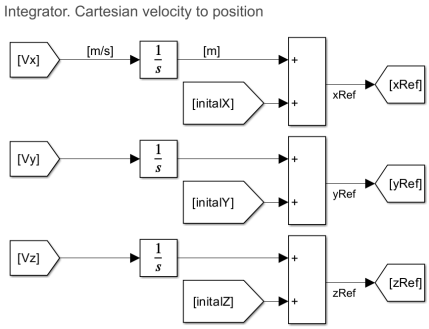

Figure 2.31: Cartesian {s} Velocity to Position

For the inverse kinematics, the inputs are Cartesian coordinates. To be able to control the Cartesian velocity instead of the position, a numerical integrator is implemented in the simulation. Thus, the initial position and actual velocity is given, and the actual reference signal xRef, yRef, and zRef are calculated as presented in Figure 2.31. The position reference is then used as input for the inverse kinematics block.

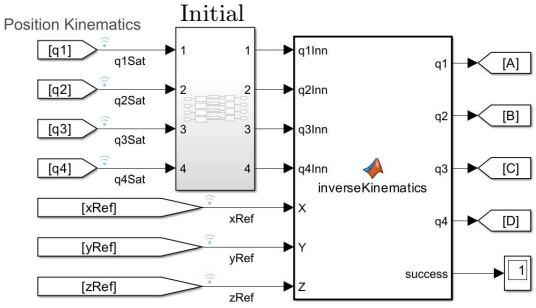

Figure 2.32: Position Kinematics Blocks

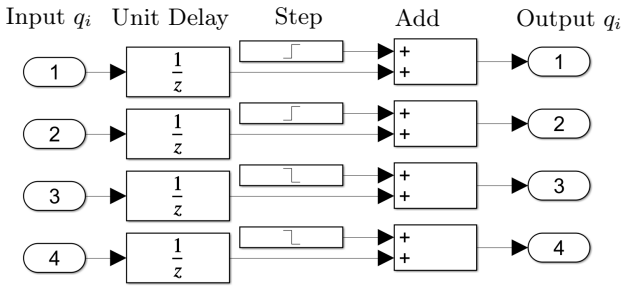

Figure 2.33: Initial Block

Figure 2.32 shows the implementation of the function inside the Simulink environment. The inputs q_1 - q_4 comes from the actual joint position in the mechanical model. The inside if the Initial block is presented in Figure 2.33. The use of this block is to give the kinematic block some data for the first iteration, when the sensor feedback is not there yet. Therefore there are step functions, starting with initial conditions, switching to zero after the first iteration. In other words, after the first iteration, it is the sensor feedback from all joints, representing the initial guess for the inverse kinematic solver. The other inputs for the inverseKinematics block are xRef, yRef, and zRef, being the desired locations in the Cartesian space {s}. The outputs q_1, q_2, q_3, and q_4 are the joint positions sent to the mechanical model. The success output is returning true or false, depending if there is a solution found for the given number of iterations. The code inside the block is presented in appendix B.7, and is using the function IKinSpace() from the "Modern Robotics" book. Because of the actuator redundancy, there are 4 unknowns and 3 equations. This is something the Newton Raphson solver does not handle. However, the IKinSpace() function uses a Newton Raphson iterative method, with the pseudoinverse of the Jacobian. The pseudoinverse is able to find the least square solution to a inverse Jacobian with rank 5 and 6 joints (including the dummy joints). In other words, by numerically manipulate q_1 - q_6 to solve the 5 equations, the actuator redundancy is solved.

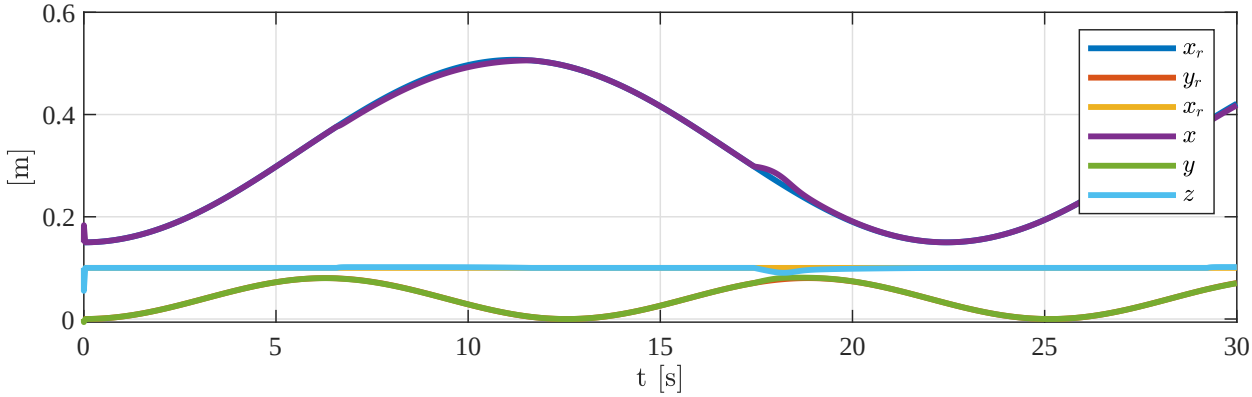

Figure 2.34: Inverse Kinematics Cartesian {s} End Effector Position

Figure 2.34 shows the Cartesian position input reference x_r, y_r, z_r plotted together with the actual position from the mechanical model x, y, and z. Some small error exists around 18 s.

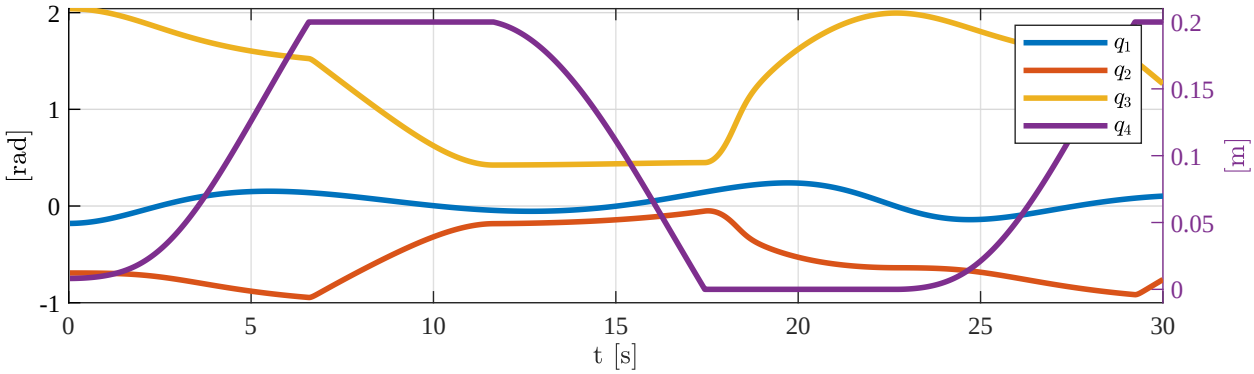

Figure 2.35: Inverse Kinematics Joint Position

Figure 2.35 is presenting the joint positions and Figure 2.36 is illustrating some corresponding screen captures of the crane simulation. From t = 0 [s] to t= 7 [s] there is small movement for the revolute joints q_1 - q_3. However, at t = 7 [s] q_4 is fully extended at 0.2 [m]. Then, because the initial guess of for q_4 is 0.2 [m] for every iteration, the inverse kinematics starts to find solutions by changing the values of q_2 and q_3 more drastically. After 12 [s] the Cartesian reference x_r is decreasing, meaning the reference is moving towards the column of the crane. Note that again, there is almost no movement for q_1 - q_3, before q_4 again stops at the end position q_4= 0 [m]. Then joint q_2 and q_3 needs to change value fast to be able to stay in their desired path. This is causing the small deviation from the Cartesian positions above. Also looking at the visualization in Figure 2.36 it seems like the inverse position kinematics always gives a solution using q_4, if possible.

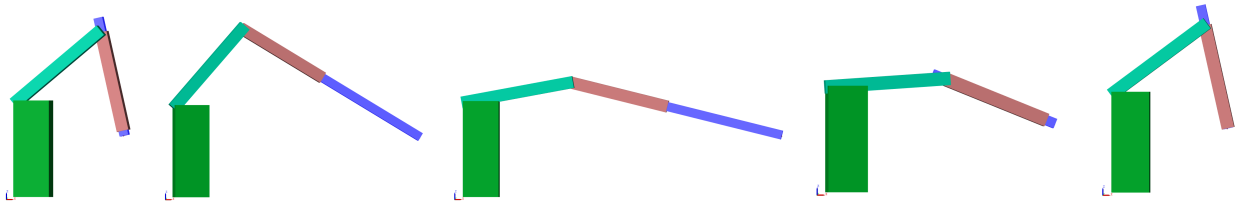

Figure 2.36: Inverse Kinematics Visualized

2.9.5 Actuator Redundancy Concept Idea

Looking at the result from the inverse position kinematics, the actuator redundancy reveals weaknesses. Because there is 3 equations, and 4 unknowns in the inverse kinematics problem, the numerical iterative process is choosing the closest solution to the initial guess. For this actual problem it seems easier for the kinematics to find a solution by changing the prismatic joint q_4.

Anyways, there are two obvious ways to solve this. Solution one: Remove one DOF/Actuator, for example by locking q_4 in a fixed position and reducing the DOF to 3 DOF. This is tested in Simulink and works perfectly. The only drawback is the workspace/reach is smaller by removing one joint. Solution two: Adding one controllable variable such as force/moment/etc to be able to solve the equations.

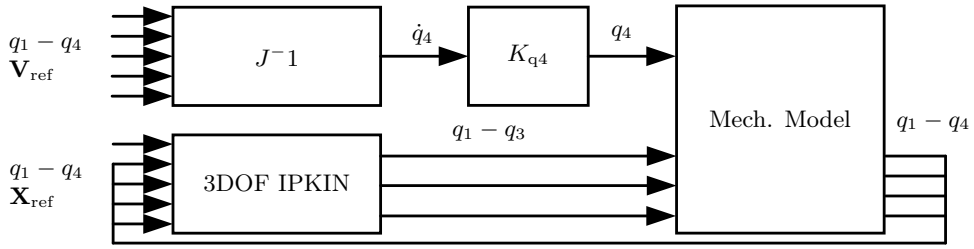

Figure 2.37: Combined Kinematics Concept

The solution proposal is based on solution one, to reduce to 3 DOF position kinematics without actuator redundancy. The concept is presented in Figure 2.37, where $\mathbf{V}_{\text{ref}}$ is the Cartesian velocity, and $\mathbf{X}_{\text{ref}}$ is the Cartesian position reference. To not get limited reach, some algorithm is going to control q_4 separately in parallel. The 3 DOF inverse kinematics will consider the crane a 3 DOF, and solve q_1 - q_3 based on **X**ref and "constant" length q_4. The next time step, the new "constant" length q_4 will be collected from the sensor, and q_1 - q_3 are found again based on the current q_4 and **X**ref.

In parallel, q_4 is controlled continuously. q_4 could be controlled based on different algorithms. However, using the Jacobian matrix, to find the joint velocity $\dot{q}_4$ seems like good idea to try. The joint velocities are calculated based on the actual joint configuration, and the desired Cartesian velocity $\mathbf{V}_{\text{ref}}$, so q_4 should move out when the velocity reference is moving away from the crane column, and vice versa. Using a gain K_{q4} on $\dot{q}_4$ gives a manipulation opportunity, and the desired q_4 is entered to the mechanical model. By using this method, the end effector should be able to follow the position reference, even with a random input for q_4, as long as the q_4 is long enough to reach the desired position. However, the solution opens for a tunable q_4, where different tests can be done, depending on the application, and K_{q4} can be tuned to fit that application. The following simulation uses the combined kinematics concept visualized in Figure 2.37.

2.9.6 Simulate Combined Kinematics Solution Without Actuator Redundancy

The combined kinematics solution is simulated and presented. Following the same path as before, the results look promising. Also, now the K_{q4} gain, makes it possible to adjust the contribution of the prismatic joint. This can also be tuned to fit the application, later when this is implemented in the IPC.

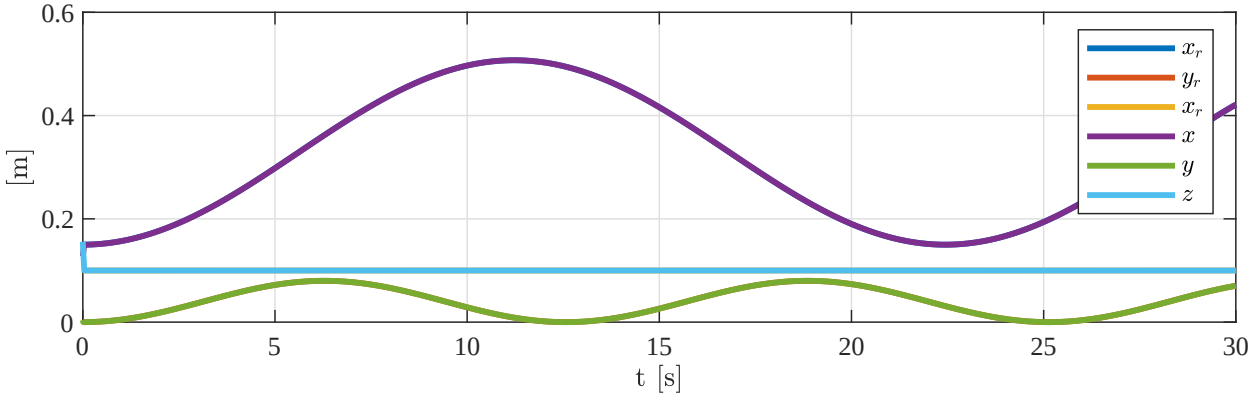

Figure 2.38: Combined Inverse Kinematics Cartesian {s} End Effector Position

The followed reference x_r, y_r, and z_r and the actual position x, y, and z is presented in Figure 2.38.

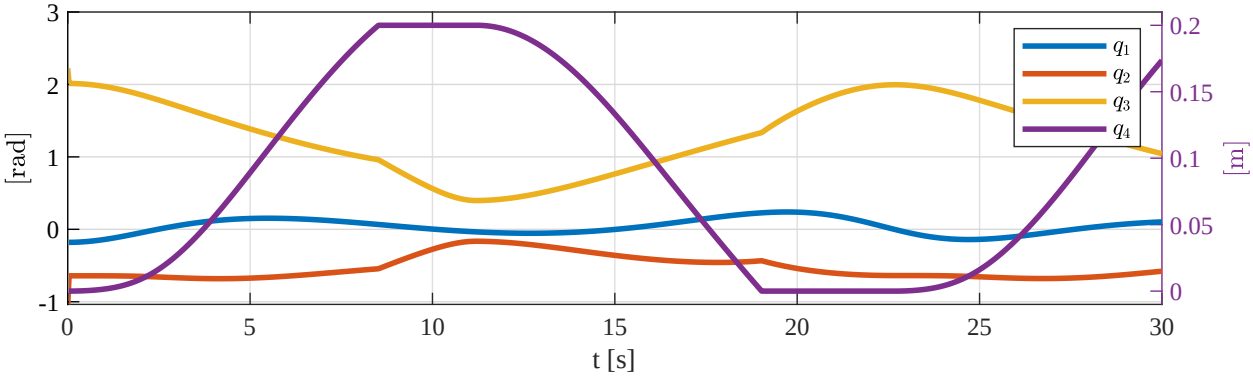

Figure 2.39: Combined Inverse Kinematics Joint Space

The joint positions are presented in Figure 2.39 and the visualization of the robotic manipulator is presented in Figure 2.40. Here, the velocity signal is working for the telescope, and we can see the 3 DOF kinematics starting the motion of the 3 revolute joints at the same time. The joint angles have less sharp changes in motion than the example above, and the crane visualization looks more natural.

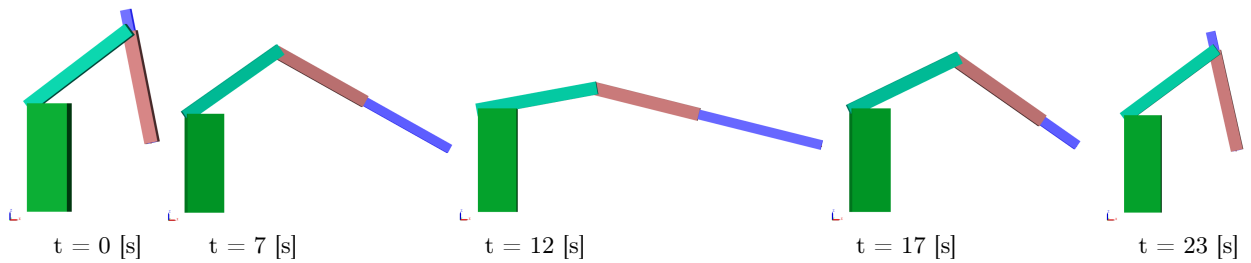

Figure 2.40: Combined Inverse Kinematics Visualization

Several different simulations where tested with different gains K_{q4}. With $K_{q4} = 0$, q_4 was locked in current position, and the end effector followed the reference until the reach was to low. With a high K_{q4}, the telescope was very active, reaching maximum or minimum q_4 fast, depending on the direction of movement. Still the end effector followed the reference $\mathbf{X}$ref, as long as it was within the reach. The results presented here was with a gain tuned to be able to follow a $\mathbf{X}$ref moving close and far away from the crane column, and the participation of the joints $q_1 - q_4$ looks reasonable.

2.9.7 Simulate Including Heave

The revolute joint used to tilt the crane, to simulate wave disturbance. Now, the ground ground position for the reference signal, and the sensor reference is moved to the frame coordinate {f}. The same kinematic solution is used as in the combined example above, except, the reference xRef, yRef, and zRef is changed to the **r1** vector explained in the Frame Coordinate to Crane Coordinate section above. The xRef, yRef, and xRef signal is moved to the input xToolpointCraneFrame in the Getr3() function block, presented in section 2.9.2.

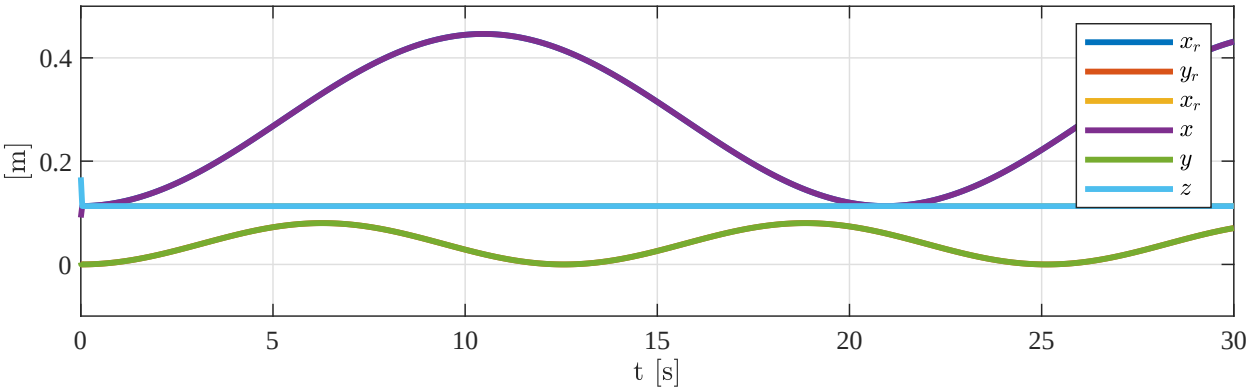

Figure 2.41: Heave Compensated Simulation Cartesian {f} End Effector Position

The end effector reference signal and the actual end effector position is plotted in Figure 2.41.

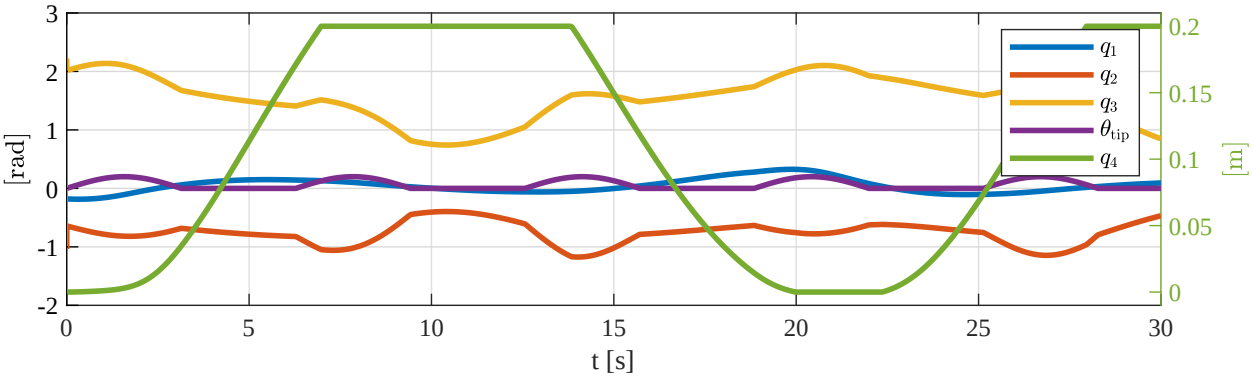

Figure 2.42: Heave Compensated Simulation Joint Space With Tipping Disturbance θ_{tip}

Figure 2.42 shows all the joint angles. Also included in this plot is θ_{tip}, which is the tipping angle cased by the disturbance tilting the crane. It is easy to see that the joints q_1 - q_3 needs to compensate for the disturbance, by their reactions when the disturbance is active. It is also clear from this plot to see that q_4 is open loop controlled, and "does not care" about the changes, but lets the 3 other joints take care of it with their position kinematics.

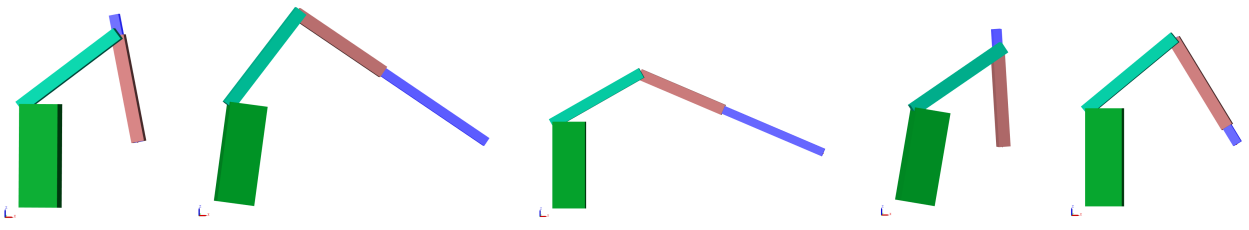

Figure 2.43: Heave Compensated Simulation Visualized

Figure 2.43 presents some visualization of the active crane motion with disturbance.

Chapter 3

Design

The design chapter includes the CAD design of the demo model, including frame design and sensor attachment. The complete CAD model can be downloaded from the public repository on github [26]. The sensors, boards, buttons and DIN rail, among other things, are found online typical as .stp files. This is practical for the design to be able to insert the parts in the assembly for implementation. The url for all the parts found online is presented in the readme file of the CAD folder in the github repository [26].

3.1 Reverse Engineering of Crane Hardware

The crane is delivered from www.leimbach-modellbau.de. Because the crane was developed several years ago, there is no existing CAD model of the crane. Therefore, one part of this project is to reverse engineer the crane to create a rough CAD model to use for visualization and development of the project.

To model the parts in the crane, the parts were clamped to a paper as and contours were made with a 0.5 mm pencil, and hole marks with a pen. The papers could then be scanned and imported to Solidworks. The original hand drawings are added in appendix C.1.

When the picture is imported to Solidworks, a reference distance needs to be set, and the picture is scaled, so it is representing real size. For the purpose of this project this method was considered sufficient because all the joints and actuators of this model crane does not have bearings, and all joints have some slack, so ± 0.5 mm from the drawing would not have critical consequences for this project.

The most critical dimensions are the distance between each joint. These distances were measured using a clipper and double-checked against the imported hand drawing.

All the crane parts reproduced and assembled in Solidworks are presented in Figure 3.1. These parts are helpful for development and represent the kinematics of the crane.

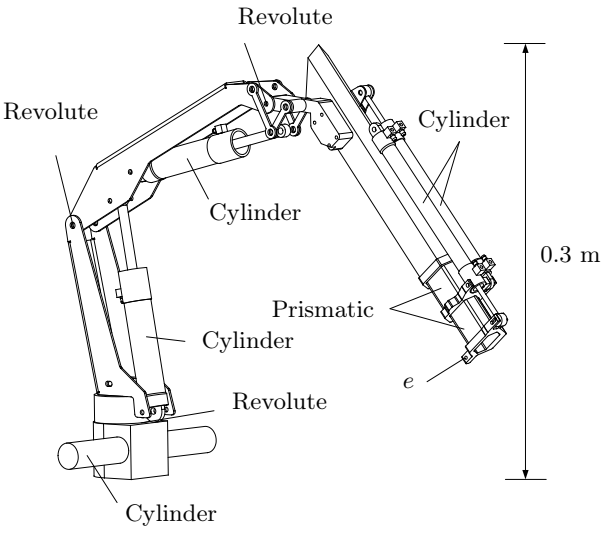

Figure 3.1: Simplified 3D Model

Pistons and cylinders were measured with a caliper. The Solidworks assemblies are simplified to not include screws, washers etc. However, the CAD model is helpful for the rest of the design. To be able to place components like microcontrollers, power supply, hydraulic component and sensors, it is helpful to have a CAD model. Because the model is mainly to design, and see that all the components needed fit in the frame, some component are modeled roughly. For instance, the control box does not look exactly as in the CAD model. However, the outer dimensions are accurate to be able to make room for it in the design. This goes for most of the components in the assembly.

3.2 Frame Design

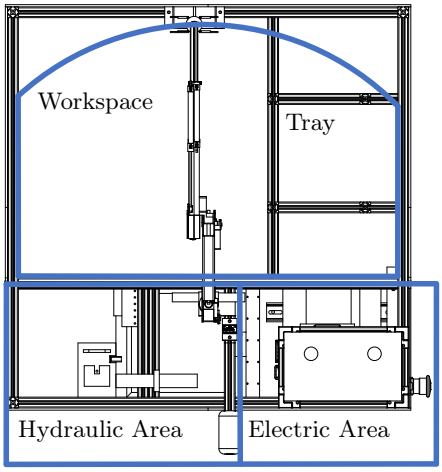

Figure 3.2: Different Sections of Demo Model

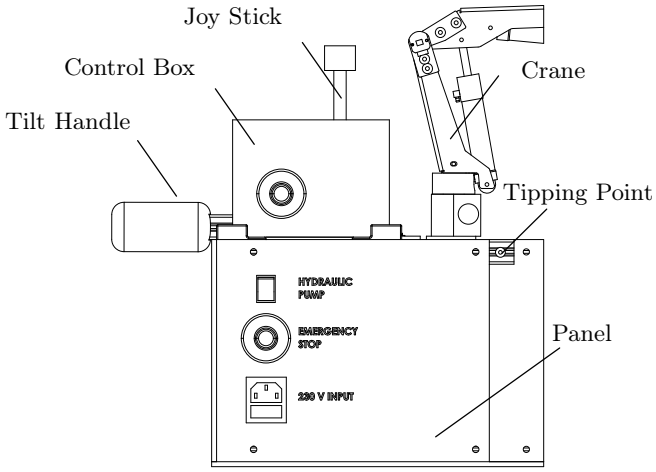

Figure 3.3: Side View Demo Model

Figure 3.2 shows the crane and frame assembly seen from above. The workspace is defined to be inside the area marked as workspace. The timber tray, marked as Tray in the figure is also inside the area. The idea behind this is that some small pieces of timber are added inside the area, and students or others can practice to control the crane by loading the pieces of wood inside the tray. The crane could theoretically reach outside the workspace, on the left and right side, but this is restricted in software.

The area including the hydraulic system and the electronics is separated as illustrated in the figure. Of course, some components, like sensors, and servo motors actuating the hydraulic valves, needs to be inside the hydraulic area, but the idea is to keep potential hydraulic leakage as much as possible away from electronics.

Figure 3.3 shows the button panel. Here, the 230 [V] supply is connected directly to power up the complete demo model. The emergency switch cuts the 230 [V] supply if anything goes wrong. The button marked with Hydraulic Pump, is directly feeding the 12 [V] motor supplying the hydraulic pump. The control box with joysticks is mounted on the top of the frame.

The tilt handle is directly connected to the crane itself. If the handle is pulled, the crane assembly is rotating around the tipping point marked in the figure.

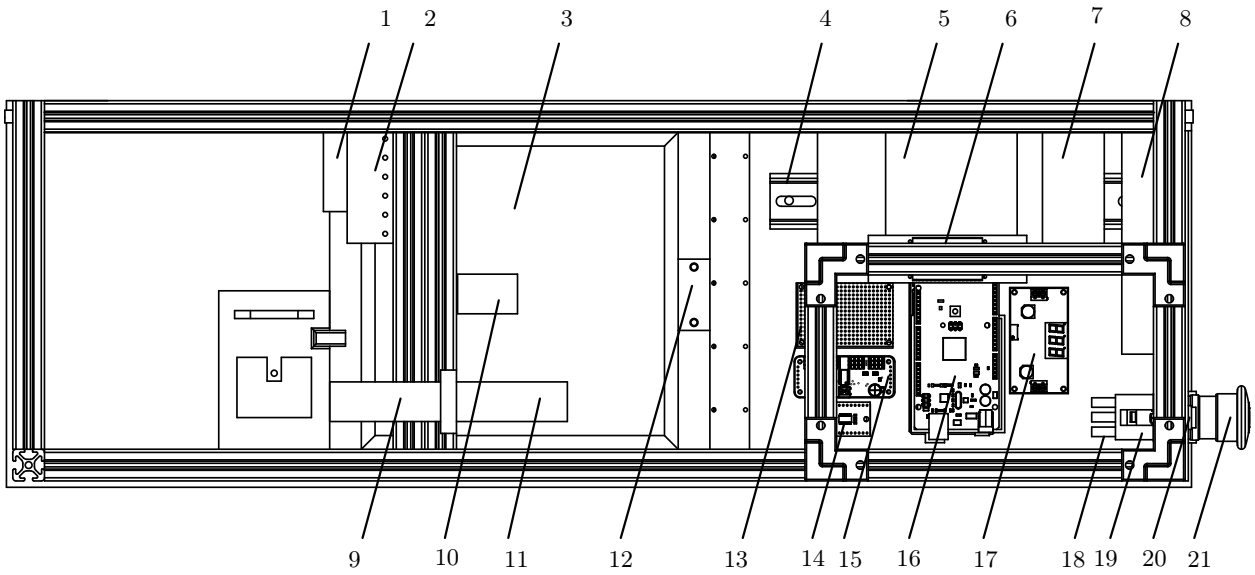

Figure 3.4: Electric and Hydraulic Components

Tag	**Explanation**
1	Servo Motors For Hydraulic Valves
2	Hydraulic Valves
3	Hydraulic Spill Tray
4	DIN Rail For IPC and Power Supply
5	Beckhoff IPC PC
6	24 [V] to 5 [V] Converter
7	Beckhoff 24 [V] Power Supply
8	12 [V] Power Supply
9	12 [V] Motor
10	Filter Hydraulic System
11	Hydraulic Pump
12	Cable Clamp
13	Veroboard (Power and I2C Distributor)
14	I2C Multiplexer TCA9548A For Encoders
15	PWM Distributor PCA9685 for Servo Motors
16	2 x Arduino and 2 x EasyCat Shield (stacked)
17	12 [V] to 5 [V] converter
18	230 [V] Input Power
19	Emergency Switch Backside
20	12 [V] Motor Switch
21	Emergency Switch

Table 3.1: Explanation to Figure 3.4

Figure 3.4 takes a closer look at the hydraulic and electric department. The corresponding Table 3.1 links all the numbers to their component.

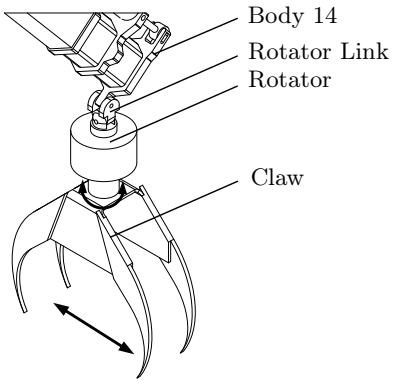

Figure 3.5: Claw Illustration

Figure 3.5 shows a simplified CAD version of the rotator and claw. Both the rotator and claw are delivered from Leimbach Modellbau and are hydraulic actuated. The rotator is connected to body 14 through the rotator link. The rotator link is creating a double revolute joint, making sure the rotator can swing freely in all directions.

The rotator is a semi rotary hydraulic actuator, meaning the rotation is restricted to a certain angle in each direction. The claw is connected directly to the rotator, and is actuated by a double acting hydraulic cylinder, opening and closing the claw.

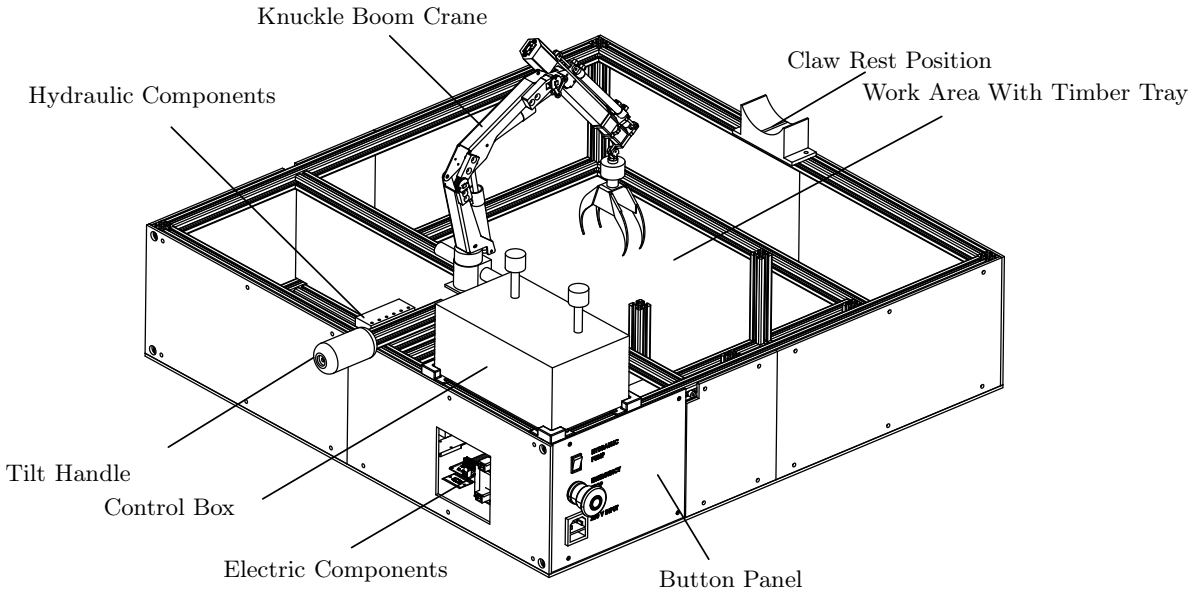

Figure 3.6: Demo Crane Assembly Illustration

Figure 3.6 shows the complete crane and frame assembly CAD model.

3.3 Sensors Attachment

As specified in the problem statement, one of the tasks, is to attach appropriate sized sensors to the crane. More details about each sensor is presented in the hardware chapter. However the sensors used on this demo model are presented in Table 3.2.

Sensor	Explanation
MPU9250	6 DOF Accelerometer and Gyro
AS5600	Magnetic Encoder
VL53l0X	Laser Time of Flight (TOF) Distance Sensor

Table 3.2: Sensors

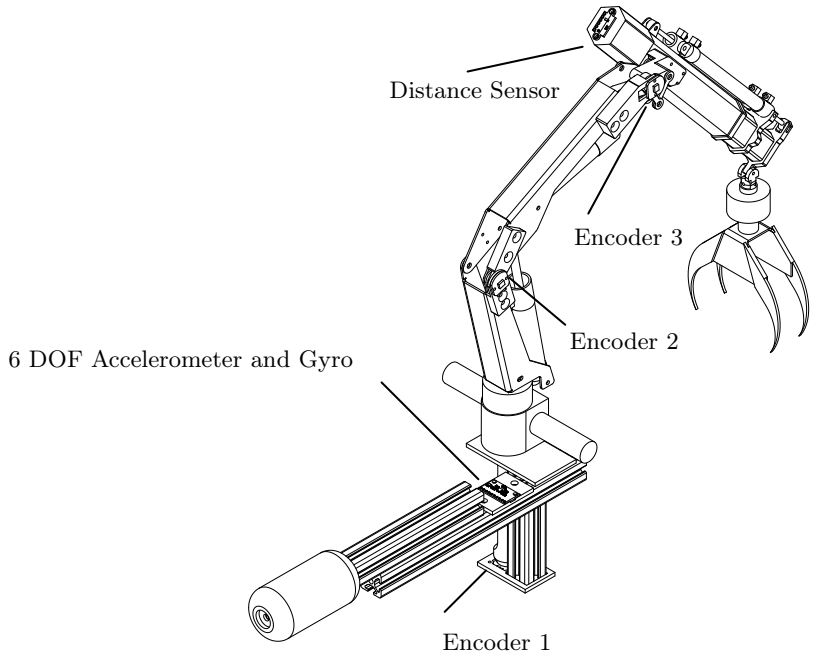

Figure 3.7: All Sensors Attached to the Crane

All the sensors used on the crane assembly are presented in Figure 3.7. Encoder 1 is measuring the the angle of rotation for joint q_1. The MPU9250 can measure location and orientation in all three coordinates, x, y, z. However, for this assembly the sensor is only used to measure the tipping angle θ_{tip} presented in the multibody dynamics chapter. Encoder 2, and encoder 3 are used for q_2 and q_3 respectively. The distance sensor is a time of flight (TOF) sensor, and is used to find the distance q_4.

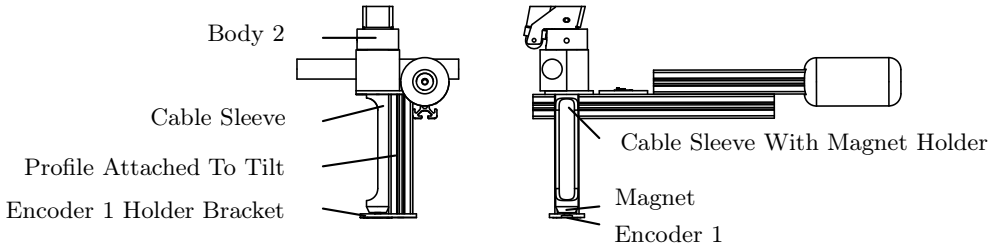

Figure 3.8: Encoder 1

A closer look at encoder 1 shows the modifications made on the crane to be able to measure the angle, with the least possible compromise on the reach of the crane. From the building manual of the knuckle boom crane, it is presented that all the hydraulic hoses needs to go through the center of Body 2 in Figure 3.9. The challenge designing the sensor attachment, is to keep the space for the hydraulic hoses, and at the same time be able to add cables for the sensors Encoder 2, Encoder 3 and the distance sensor. The cable sleeve with magnet holder presented in the figure, is a simple sleeve with a large half moon like opening to exit the hoses and cables. The sleeve is pressed and glued to body 2, and rotates with it. The profile is attached to the tilt, and makes it possible to attach the non rotating encoder 1 holder, under the sleeve. The magnet, witch is included with the magnetic encoder, is pressed in to the bottom of the sleeve. With this assembly, encoder 1, is reading the direct angle between the fixed tilt, and body 2.

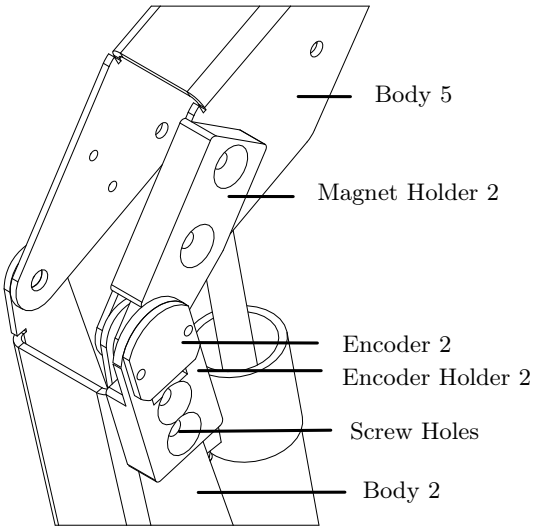

Figure 3.9: Encoder 2

Figure 3.9 shows encoder 2 with the attachments. For this, a slightly different encoder compared to encoder 1 is used. This is explained more in the hardware chapter. The magnet holder is attached to body 5. Two holes are drilled and threaded in body 5, to secure the magnet holder. The magnet is pressed and glued to the magnet holder. Therefore, the magnet is always following body 5. The encoder holder 2 is fixed with screws to body 2. The encoder is screwed to this bracket, so the encoder is always following body 2 and the magnet is following body 5. Thus, the angle between them can be measured.

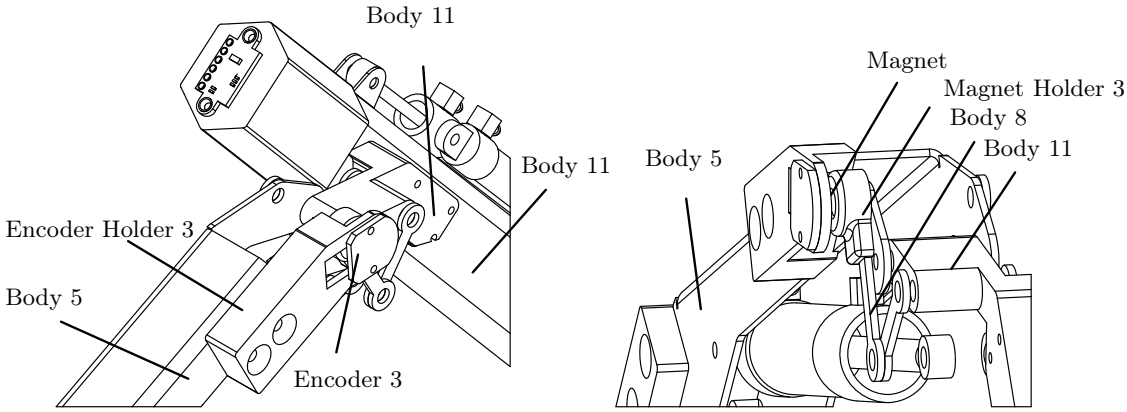

Figure 3.10: Encoder 3

Encoder 3 is presented in Figure 3.10. The idea here is the same as for encoder 1 and 2. However, the magnet is fixed differently here. The desired angle is the angle between body 5 and body 11, but body 8 is blocking the possibility to attach the encoder directly. Therefore the magnet holder is attached to body 8, and the angle between body 5 and body 11 is calculated based on the angle between body 5 and body 8.

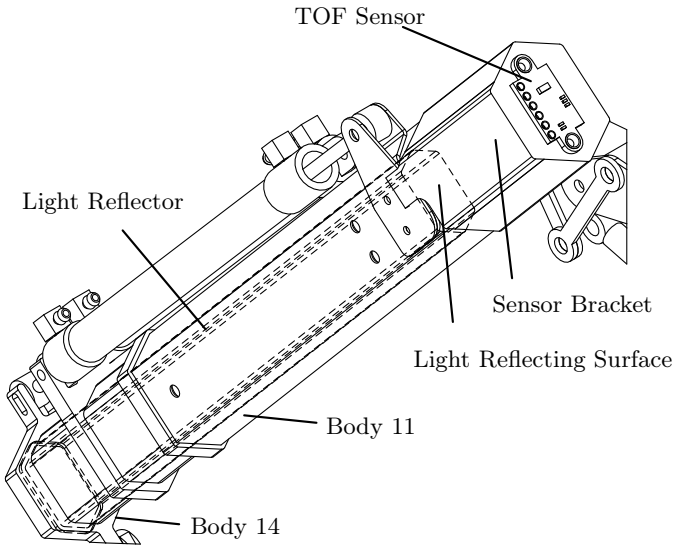

Figure 3.11: TOF Sensor Attachement

The purpose of the TOF sensor is to measure the distance between body 14 and body 11, corresponding to the length q_4. The idea is that the light of the TOF sensor is inside the hollow telescope, and reflect on a light reflector. More details and experiments are presented on this later in this report.

Figure 3.11 shows the telescope assembly with the mounted sensor. The sensor measuring the distance between body 14 and body 11 is mounted to body 11 through a sensor bracket. The light emitter and reciver of the TOF sensor is not mounted in the center of the TOF sensor module. Therefore, the sensor is attached with corresponding offset from center of the sensor bracket, so the light beam is in the center of the telescope. A long light reflector is mounted inside of body 14. The end of this, is marked in the figure as light reflecting surface. This is where the light from the sensor is reflecting and calculate the distance between body 11 and body 14 based on this.

Chapter 4

Hardware

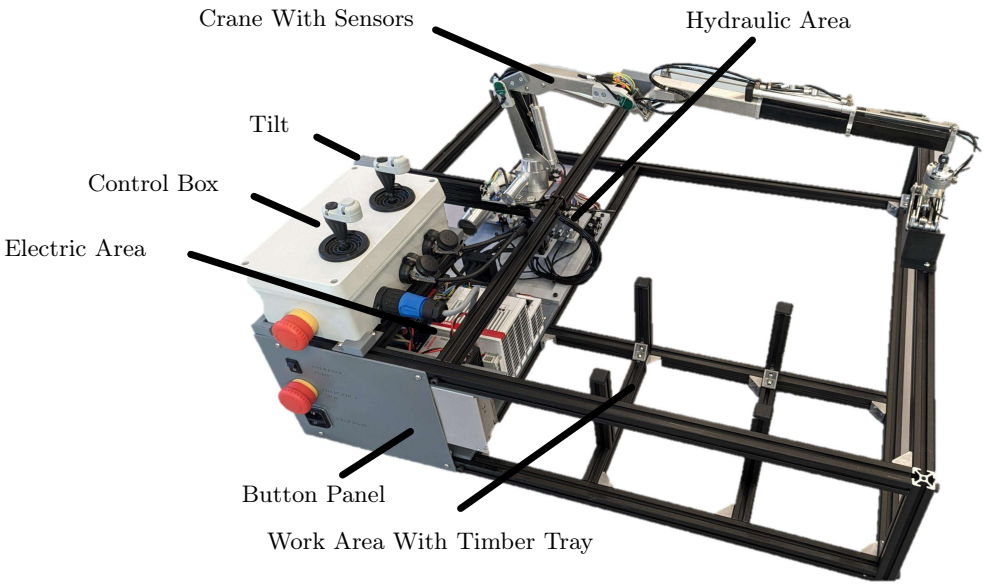

Figure 4.1: Crane Assembly

Now that the design of the demo model is presented, this chapter is presenting the hardware of the model. In other words, which different components are used and how are they connected together. Figure 4.1 shows the assembled model, without cover panels mounted.

4.1 Crane and Hydraulics from Leimbach Modellbau

4.1.1 Crane

The crane is delivered from Leimbach Modellbau in parts, together with a build manual. The crane part number is 09401, and all the documentation for the assembly of the crane is found on the link presented in reference: [20]. Press the button "Anleitungen/Dokumentation" to get the list of documents, and then open "Bauanleitung Klappladekran gesamt (09401+09402+09403)" to see the work instruction to assemble the crane.

The rotator and the claw are normally designed for another crane called the Langholz-Ladekran,

and the assembly instructions can be found in the following reference [21]. From page 36 the rotator and claw assembly is presented. The rotor can rotate approximately 270 degrees in total.

4.1.2 Hydraulics

Hydraulics transports the power through fluid, and in this crane the all the actuators that provides the motion of the crane are hydraulic actuators. Because this is a model crane, the hydraulic system is very simple compared to a full size system. However, many similarities for this model crane like double acting cylinders, direct control valves (DCV), semi rotary hydraulic actuator, pump and filter. Most hydraulic systems also have a pressure relief valve (PRV) to make sure the pressure in the system is not rising above the adjusted level for the PRV. For these hydraulic components, the information found on Leimbachs web site is limited. This is understanding considering the application of use the different actuators are used for.

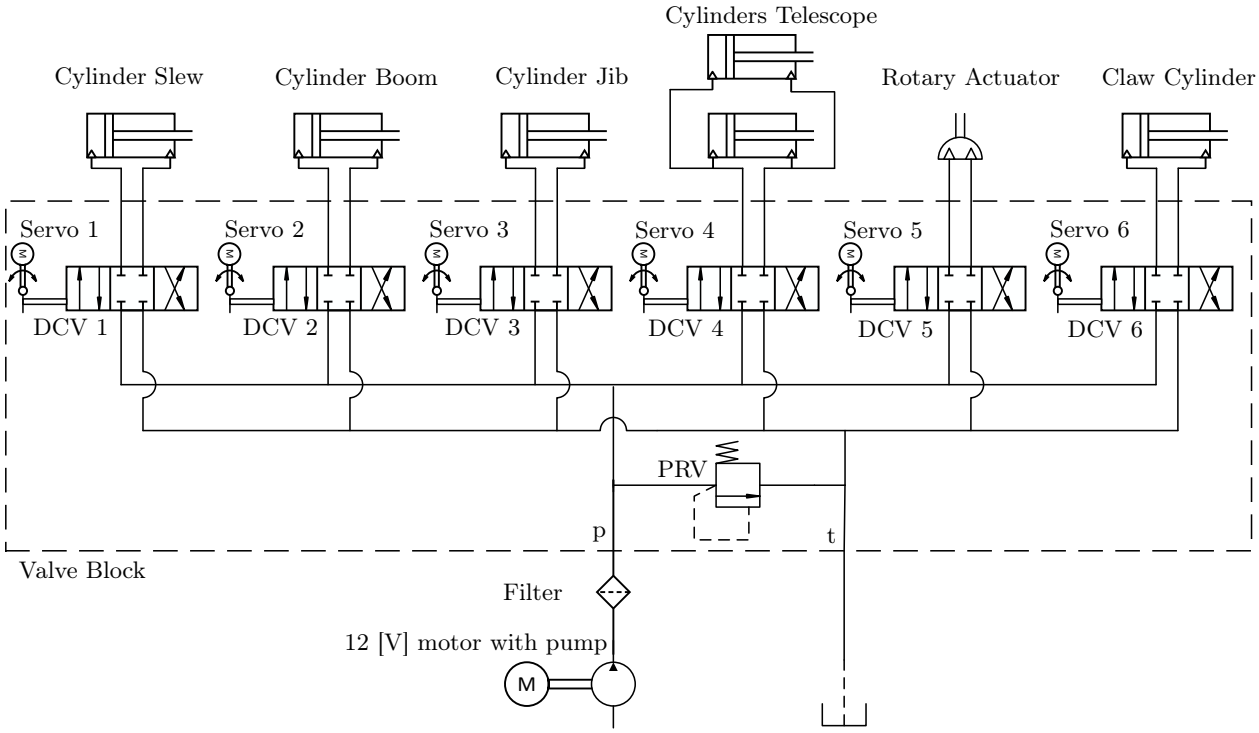

Figure 4.2: Hydraulic System

The hydraulics schematic is not presented from Leimbach, but the schematics for this assembly is included in Figure 4.2. However, the details inside the valve blocks are not completely clear, but the assumption is that there is a pressure release valve inside the valve block. Note the uncommon way of actuation used for the DCVs. Normally they are actuated either by a mechanical handle, spring, pilot pressure or/and electric spools. In this case, the valve is a mechanical actuated valve, but there is a small servo motor mounted on the mechanical wheel controlling the opening of the valve. Therefore, in the schematics each DCV is equipped with a motor attached to the mechanical arm. The hydraulic pump is driven by a 12 [V] DC motor. The motor is controlled directly from with a switch.

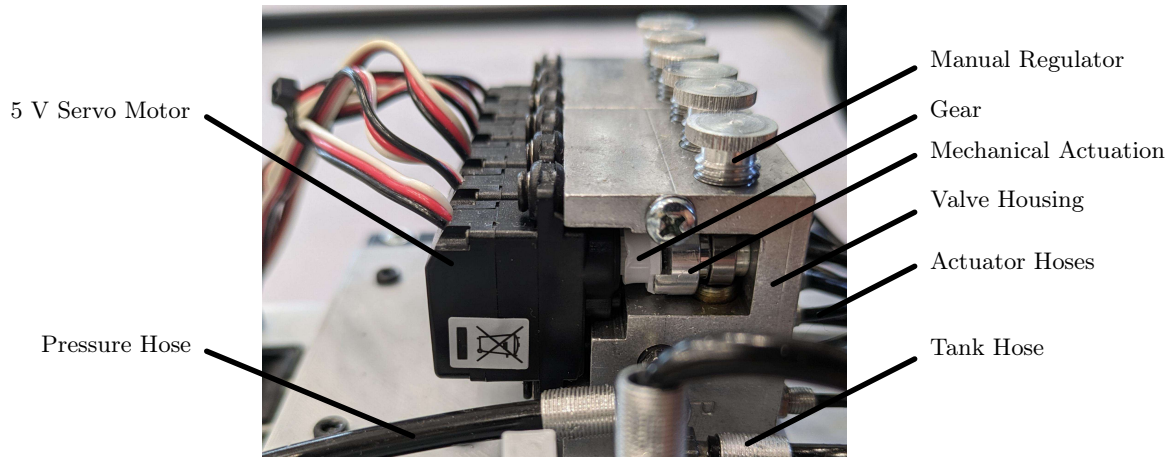

Figure 4.3: DCV

Figure 4.3 shows the rack of valves and servo motors. Here, the electric/mechanical actuation is presented in more detail.

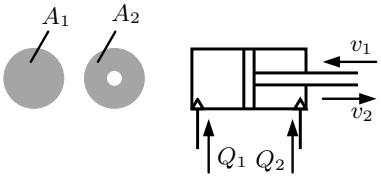

Figure 4.4: Piston and Rod

$$Q = Av \tag{4.1}$$

Figure 4.4 illustrates the difference between A_1 and A_2. Lets imagine that the pressure is constant, and Q_1 is flowing in with a certain constant rate. From the volumetric flow rate equation 4.1 the velocity is found. If the same flow is used for Q_2, it is clear that the velocity will be smaller because the area A_2 is smal ler. This is something to have in mind for the control of the actuators.

4.2 Sensors

This demo model is using different sensors. The objective of the sensors are to keep track of $q_1 - q_4$ and θ_{tip}, so that the end effector of the crane can be calculated or/and controlled based on the angles and distances. The demo model is equipped with three different sensors. Magnetic rotary encoders measuring the angle of the joints, laser time of flight sensor measuring distance of the telescope, and a 6 DOF combined accelerometer and gyroscope to measure the orientation and location of the crane itself. For this crane setup, a rotary encoder would be sufficient to measure the tilt angle, but for a offshore vessel, a accelerometer and gyroscope could be used to be able to track the position and orientation of the vessel.

From the problem statement, it is stated that appropriate sized sensors should be used. Another aspect that is drawn attention to regarding the choice of sensors is the price and availability. The AS5600, VL53l0X, and MPU9259 are all sensors that UIA uses project work in the bachelor engineering courses. Therefore the availability was good, but also the aspect of showing component on a demonstration model, that actual are used in common projects is put weight on, for the sensor choice.

4.2.1 AS5600

The AS5600 is a 12 bit programmable magnetic rotary encoder. The sensor communicates with I2C communication protocol. Several advantages are listed in the data sheet [1], such as contact-less, simple to use, high resolution, small form factor, and robust for different environments.

One disadvantage with this sensor, is the static I2C address, meaning only one of these sensors can be used on the same bus. Because of this, a I2C multiplexer is needed to communicate with the different sensors with the same address.

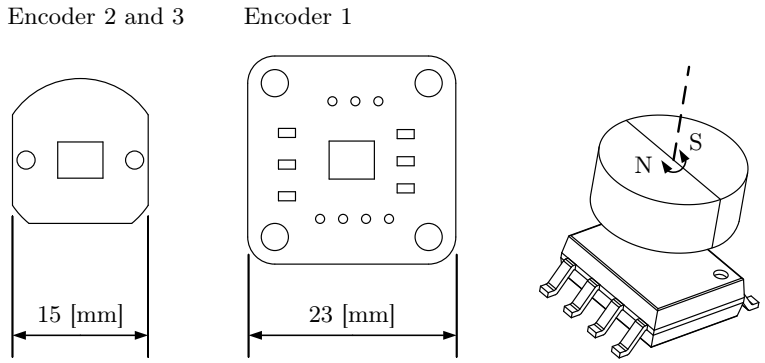

Figure 4.5: AS5600 Illustration

For this design, the size of the sensor is an important criteria. The chip of AS5600 itself is small, but different sizes of circuits boards with this sensor are available. Both types used for the demo crane are illustrated in Figure 4.5.

Figure 4.5 also shows the working principle of the chip together with a magnet. The permanent magnet, need to be split vertically, with the north and south pole as presented in the figure. Placing the magnet 0.5 - 3 [mm] away from the chip, the angle between the magnet and the chip can be read.

4.2.2 I2C Expander

TCA9548A I2C Expander/Switch is used for the rotary encoders because they all have the same I2C address. The device has eight bidirectional switches [28], meaning one encoder is physical connected to the Arduino at the time, transmitting data, before switching to the next one, and then the last one before it starts over at the first encoder again. The available clock frequency is 0 - 400 [kHz]. With this, it is possible to use multiple I2C devices with the same address, because when the master is calling the address of the sensors, the others are physically disconnected by the switch.

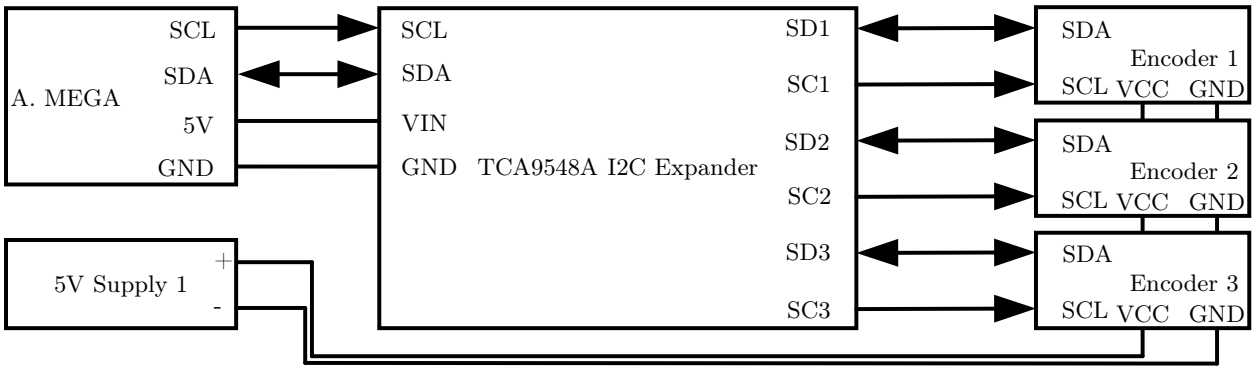

Figure 4.6: I2C Expander

Figure 4.6 shows the connections to the I2C expanded. Here, the I2C master is the Arduino Mega.

4.2.3 TOF Distance Sensor VL53l0X

VL35L0X is a distance measuring device with a range up to 2 [m] [29]. The sensor is a time of flight sensor communicating over I2C. The field of view of the sensor is 25 degrees. This is more than desired for the inside of the telescope, and there is a risk of unwanted reflections from the side walls. Some experiments are done with the VL53L0X. The experiences are based on using different light reflectors inside the telescope, actuating the hydraulic cylinders of q_4 and measure the different outcomes.

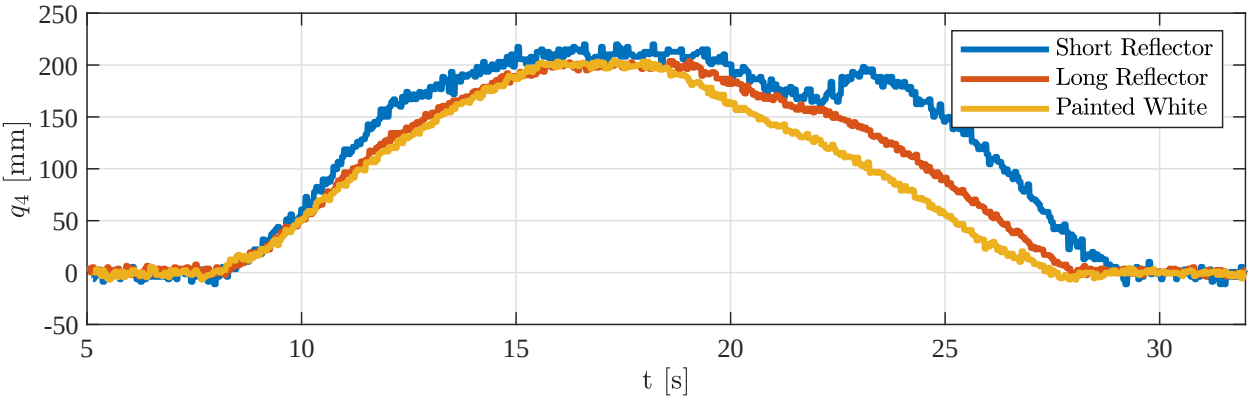

Figure 4.7: Distance Sensor Experiments

Figure 4.7 shows the measurements from three different light reflectors, with the telescope doing the same motion. From fully retracted at 0 [mm], to maximum extension at 200 [mm], to minimum extension again. Note that before the data collection started, on each different attempt, the minimum and maximum sensor value was collected, and the signal was scaled to stay between 0 and 200 [mm] based on that. This is for a better comparison of values, even with different actual length between the sensor and the reflector.

The first test, plotted in blue, was with a 3D printed plastic reflector barely going inside the telescope. Until around 22 [s], the graph looks ok. However, between 22 [s] and 25 [s] there is a jump in the distance, even if the telescope is retracting at the same velocity as before. The second test, plotted in red, is with a long light reflector entering the telescope, so the distance between the reflecting surface and the sensor is less than before. The result is much better, but test provided in the data sheet [29] states that the sensor is more accurate reflecting on white. The reflection surface is painted in white, and the result is plotted in yellow. Even better and more linear result.

4.2.4 MPU9250

The MPU9250 is a 16 bit combined gyroscope, accelerometer, and magnetometer [22]. The MPU9250 is communicating with I2C, and have several programmable features. For this application, the sensor is attached on the body of the handle, and is following the orientation of the crane. Due to the restricted motion provided by the tilt, only rotating around one axis, the only data used from the accelerometer for this application, is the angle around the tipping point θ_{tip}. The MPU9250 is connected to the veroboard, and is presented in the veroboard schematics.

4.2.5 PWM module

The PCA9685 servo driver is a module able to serve PWM signal to 16 different channels, only by using two pins from the micro controller [3]. The servo driver receives signal through I2C, external power supply, and sends PWM, and power to each servo motor. For this application the

servo module is connected to the output Arduino and all the six, 9g servo motors mounted on the hydraulic valve.

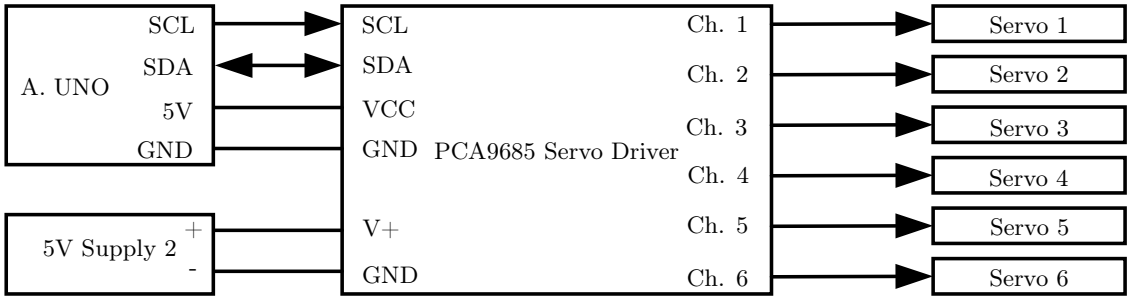

Figure 4.8: PWM Module Shematics

The schematics of the servo driver is presented in Figure 4.8. Note that the arrows from channel (Ch.) 1 - 6 represent all three cables for each servo motor. 5V, GND and PWM.

4.3 Sensor To EtherCAT Interface

Because all the sensors are communicating with I2C, and we want the crane to be controlled by a Beckhoff IPC, first we need a micro controller handling the I2C signal, and translate this to a protocol that the IPC can understand. One possibility is to translate to EtherCAT and then connect to the EtherCAT bus of the IPC.

The demo crane is equipped with two micro controller, one Arduino Mega for the sensor inputs, and one Arduino Uno for the servo motor outputs. Both controller are equiped with each their own EasyCAT shield.

4.3.1 EasyCAT Shield for Arduino

The purpose of the EasyCAT shield is to use the Arduino board as a EtherCAT slave [2]. The EasyCAT shield is plugged directly on top of the Arduino board, and communicates with the Arduino through Serial Peripheral Interface Bus (SPI). Busano.net delivers the documentation and software needed to use the shields. Configuration of the EasyCAT Shields is explained in the software chapter.

4.3.2 Arduino Output

The output controller, controlling the servo motors, is an Arduino Uno. The Arduino Uno is a famous developing board using the ATmega328P and the ATMega 16U2 prosessor [7]. The purpose of the controller is to translate the I2C data to EtherCAT, through the EasyCat shield.

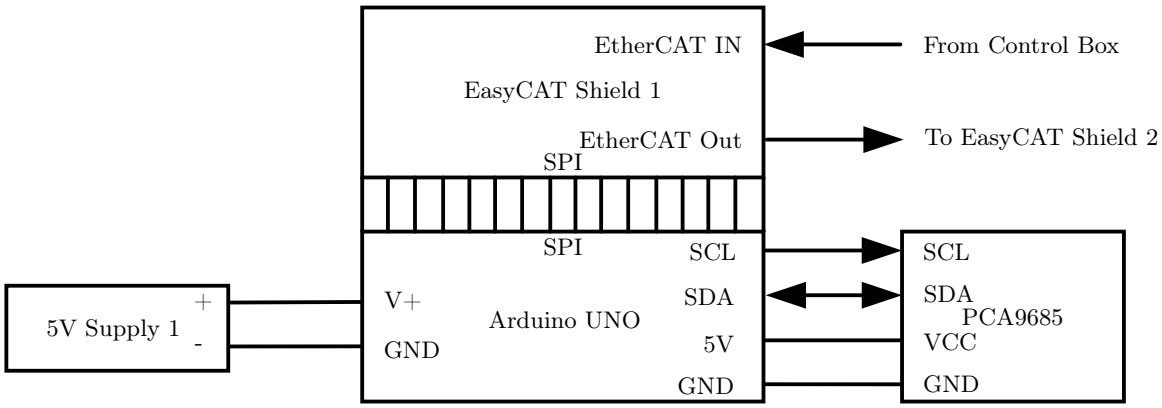

Figure 4.9: Arduino Output Shematics

Figure 4.9 shows the wire connections. The EasyCAT shield is mounted directly on top of the board, and all pins are connected.

4.3.3 Arduino Input

The Arduino Mega 2560 is a more powerful development board using ATmega2560 micro-controller [6]. The reason to separate the input and the output in to two different slaves, is higher reliability for manual crane control. Some test revealed problems with the I2C bus for the sensors. Using two different controllers, gives the flexibility to use the crane in manual mode without the sensor data, if the I2C bus crashes because of one sensor.

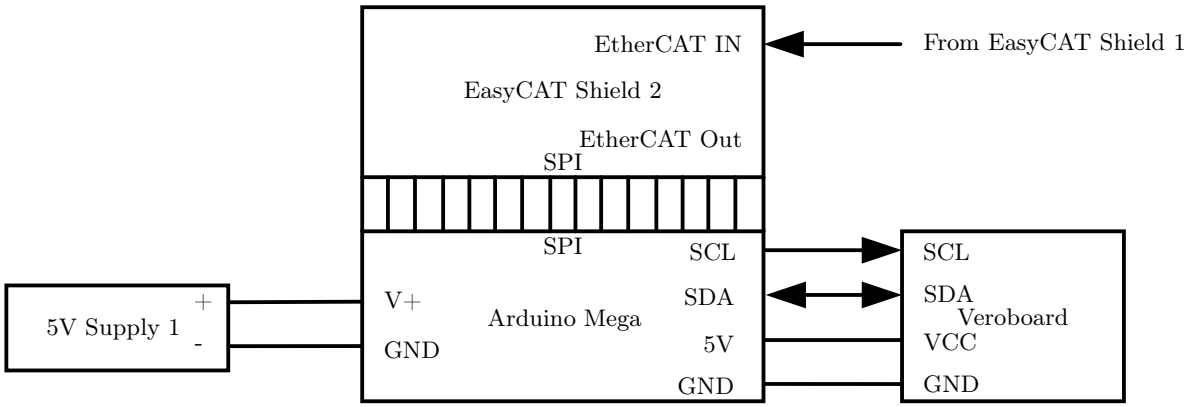

Figure 4.10: Arduino Input Shematics

Figure 4.10 shows the connections of the input controller. Note that there are multiple I2C slaves connected to this controller, thus the use of a soldered development board called Veroboard is added. The connections from the veroboard to the different sensors are presented in Figure 4.11.

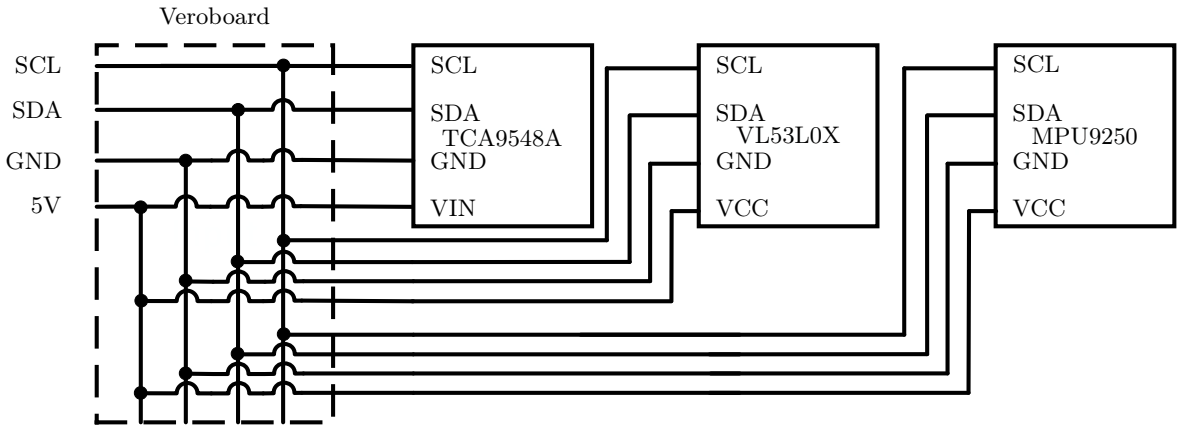

Figure 4.11: Veroboard Schematics

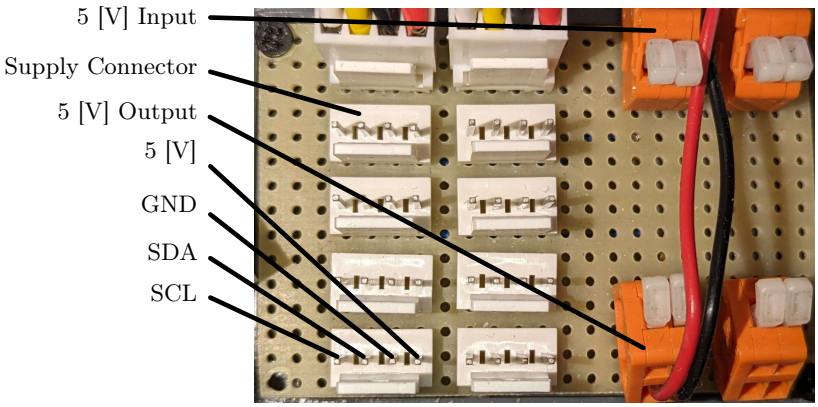

Figure 4.12: Veroboard Assembly

The Veroboard is soldered with multiple connectors with VCC, GND, SDA, and SCL as presented in Figure 4.12. This is to be able to crimp cables and use proper connectors, but still have the possibility to connect them to all of the connectors.

4.4 IPC

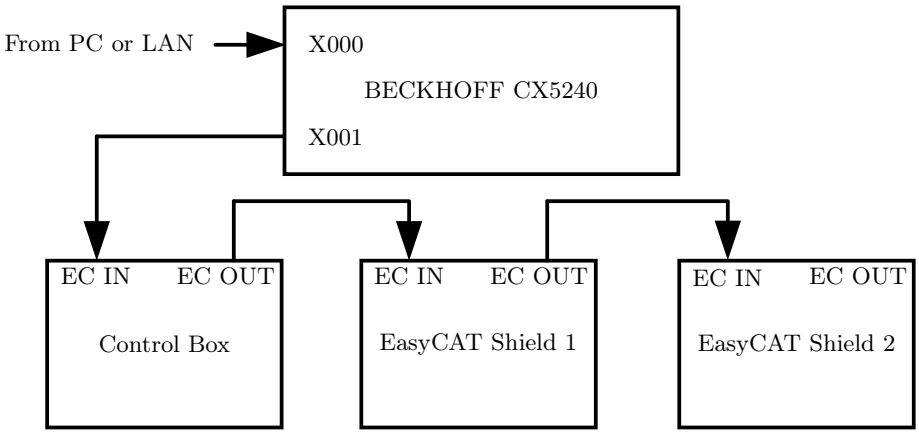

Figure 4.13: EtherCAT Connections

The top level system is the IPC with the EtherCAT slaves. They are connected together as presented in Figure 4.13. In a real life system, the connector X000 would probably be connected to the last slave, to create a circular connection with redundancy. This is not done for this demo crane, because

connector is X000 is used to connect the programming computer with the additional human machine interface (HMI), and the redundancy is not necessary for such a non critical application.

4.4.1 Beckhoff IPC

The IPC used to control the crane is a Beckhoff CX5240. This is an embedded computer with a clock speed of 1.6 [GHz] and 8 GB RAM [9].

Figure 4.14: BeckHoff CX5240

4.4.2 Control Box

The control box is connected to the IPC as presented above. The control box is consisting of two Beckhoff components. The remote EtherCAT digital output/input Beckhoff EtherCAT EK1818 [10]. This module is used to read and write the digital signals for buttons and lamps. The EK1818 is extended to read analog inputs by adding the Beckhoff EtherCAT EL3255 [11]. The analog signals are coming from the Joysticks.

Also included in the control box are joysticks, CAT6 connectors, lamps and switches.

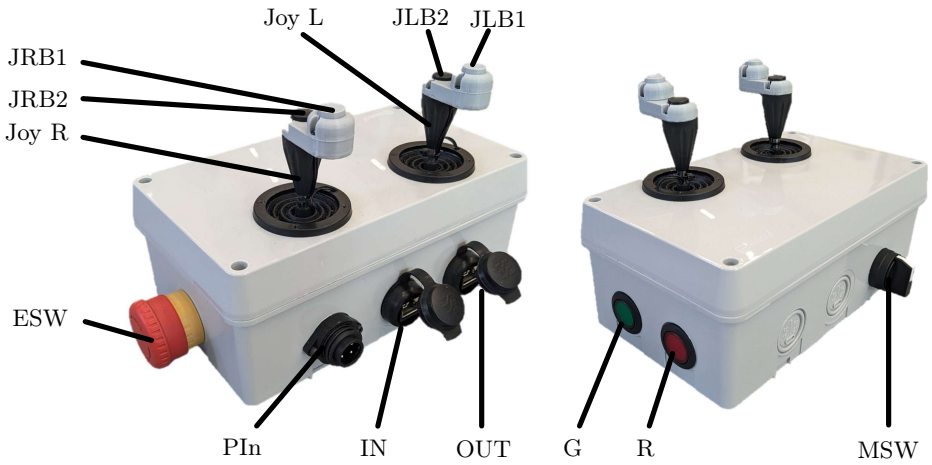

Figure 4.15: Control Box Picture

The control box is presented in Figure 4.15 and the corresponding schematics is presented in Figure 4.16. The joysticks had only one button initially, so some modifications to them were made to include one more button for each of them. The control box is built based on other EtherCAT control boxes in the UiA Mechatronics Lab, built mainly by Daniel Hagen. Therefore this box can be used to control other Beckhoff controlled cranes as well.

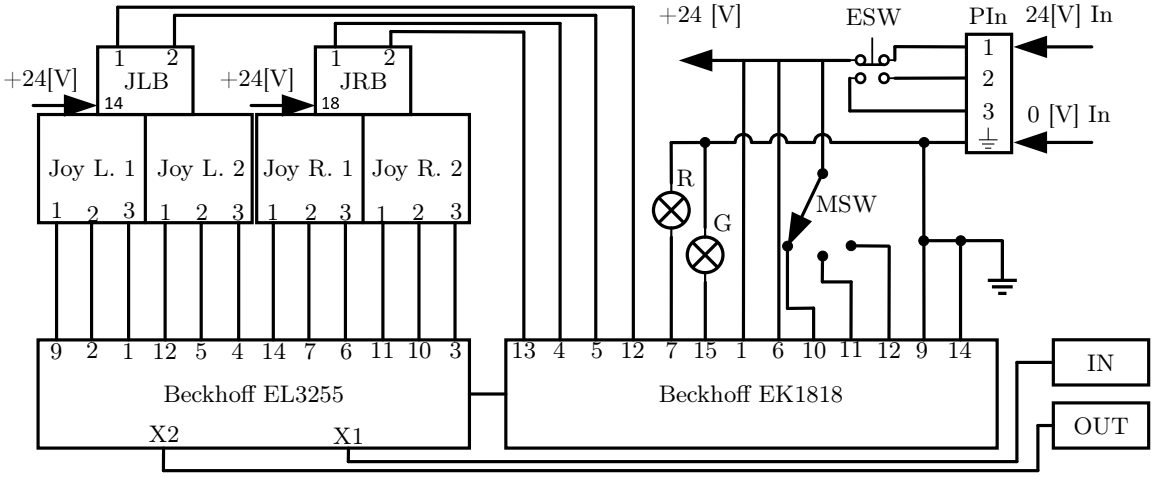

Figure 4.16: Control Box Schematic

4.5 Power Supply

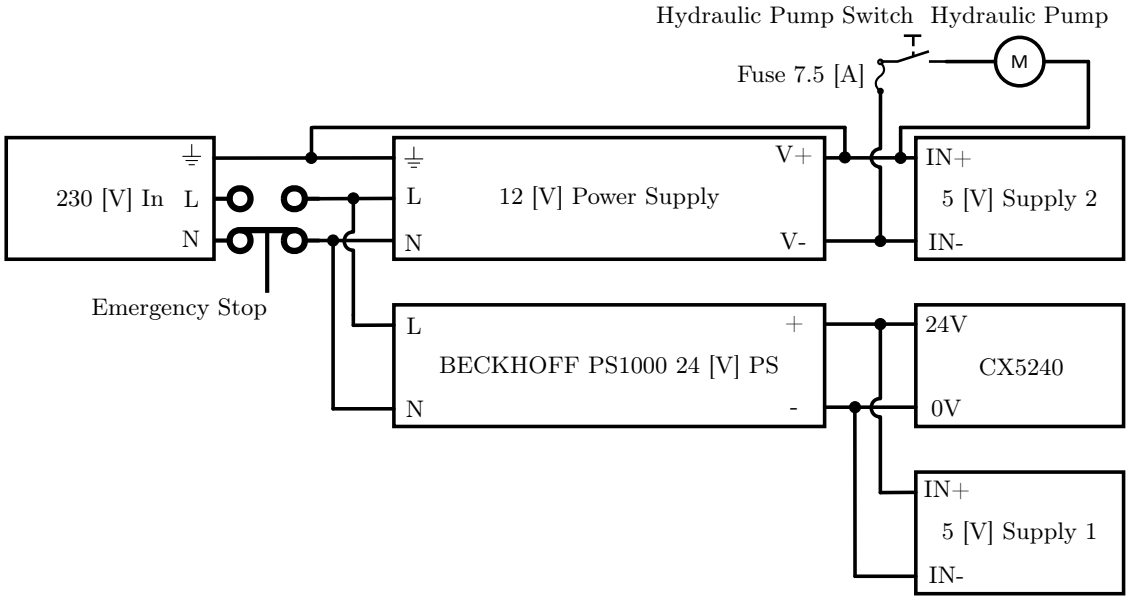

Figure 4.17: Power Supply Circuit

The power supply design is presented in Figure 4.17. There is one 230 [V] input. However several converters are used within the electrical part. First one 12 [V] power supply is connected. This is used to control the 12 [V] motor before it is converted to 5 [V] in 5 [V] supply 2, and supplies the servo motors. The 12 [V] motor causes a lot of noise, and all the motors are therefore connected to this source. All the controllers and sensors however, are connected to a different source. The PS1000 supply is serving the IPC CX5240 24 [V], before power Supply 1 is converted to 5 [V] and serving the Veroboard, that again serves the sensors and boards through a source with less noise.

Chapter 5

Control

5.1 Different Modes

The crane can be controlled in two different modes. One manual control, meaning each axis on the joystick or each button is directly connected to their actuator. The other mode is Cartesian control mode. Here, the joystick is controlling the end effector in Cartesian coordinates, using the kinematics to calculate the desired joint positions.

5.1.1 Manual Mode

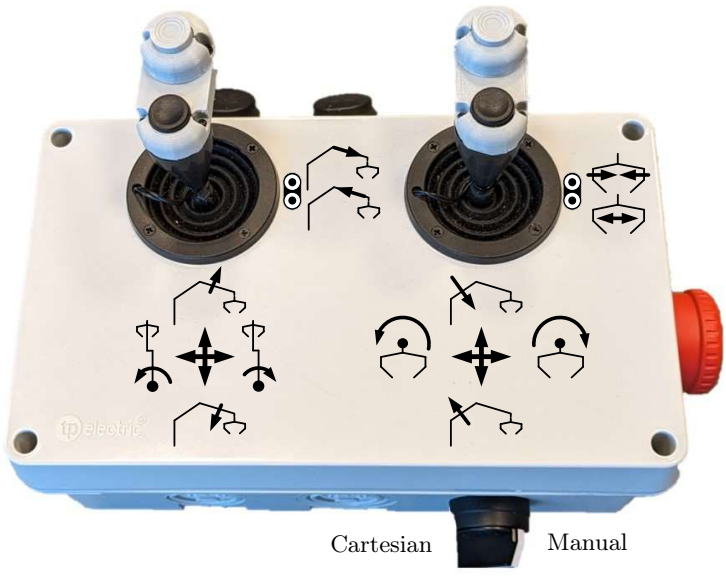

Figure 5.1: Manual Crane Control

For the manual control, the crane is controlled in joint space. Each actuator is actuated as presented in Figure 5.1. The manual mode is based on the ISO standard for excavators [32], and a real life John Deere timber Forwarder [14]. The extra buttons mounted on the joy stick are necessary to have control over all actuators.

5.1.2 Cartesian Mode

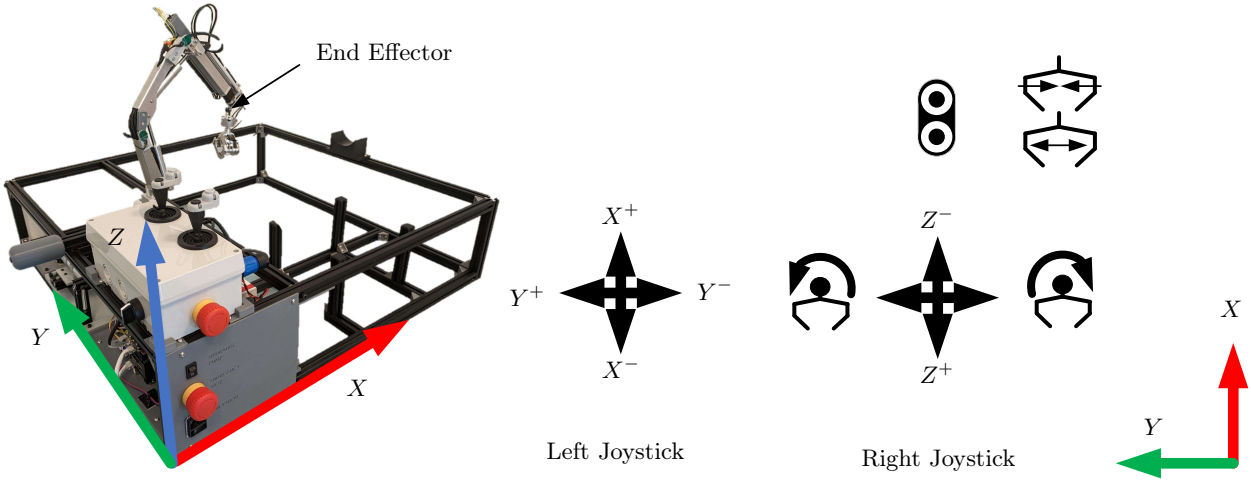

Figure 5.2: Cartesian Crane Control

For the Cartesian controller, the joysticks controls the end effector in Cartesian coordinate directly. Figure 5.2 shows how the joysticks control the end effector, presented relative to the frame. No matter what joint configuration the crane have, within the workspace, the end effector should only move along the axis actuated. For example if the left joystick is moved forward, the end effector should move parallel to the X axis. The claw rotation and clamping is outside of the kinematic representation of the crane, and is therefore controlled manually as before.

5.2 Manual Control

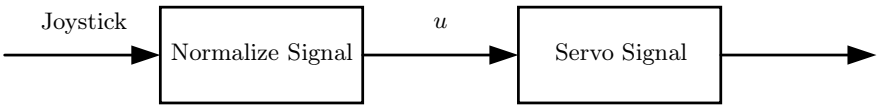

Figure 5.3: Manual Open Loop Control

The manual controller works as most manual crane controls, using analog joy sticks. Figure 5.3 shows the open loop control. This counts for all joints using the analog signal and their respective actuator. The signal from the joystick is normalized first, then scaled to fit the servo motor output.

The joystick signal comes as a analog integer value in to the IPC, and varies from 0 to around 32000, with around 16000 when the joystick in neutral position. The exact maximum, minimum, and neutral signal varies from joystick and axes, and needs to be scaled for each joystick. This signal is normalized so it is at 0, when the joystick is at the center position, - 1 at minimum and 1 at maximum.

$$u = \frac{\text{Joy Stick Raw} - \text{Joy Stick Minimum}}{\text{Joy Stick Maximum} - \text{Joy Stick Minimum}} \tag{5.1}$$

The servo is controlled with PWM signal, so the normalized signal u is translated to a PWM signal within boundaries.

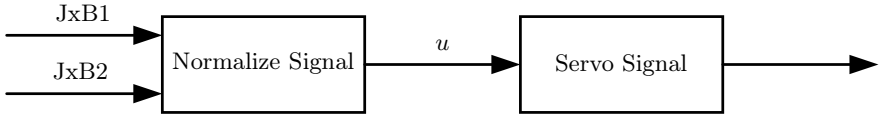

Figure 5.4: Button

For the digital buttons controlling the telescope and the claw, the output is a constant value if the button is pressed. Figure 5.4 shows the control of one of the two actuators. Here, the signal u is always 0, -1, or 1, depending on which button is pressed.

In a full size crane this signal would be ramped, to avoid oscillation. However for this small crane, and the relatively slow servo motors, this is not necessary.

5.3 Cartesian Control

For the Cartesian control, the control system is more complex. The actuator redundancy challenge is introduced in the multibody dynamics chapter, and a proposed solution is tested using a combined inverse kinematics solution with open loop velocity control of the telescope, and 3 DOF inverse position kinematics. In simulation this gave promising results, using ideal position drivers on each joint. Because of the hydraulic system, and the attached sensors some similar solution should be possible to achieve.

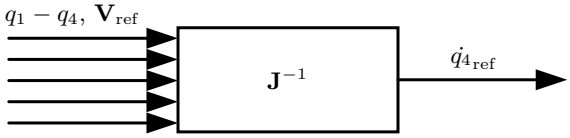

Figure 5.5: Inverse Velocity Kinematics

Firstly, Figure 5.5 shows a simplified version of the inverse velocity kinematics. Here, the actual joint position is read from the sensors and the Cartesian velocity reference is given from the joystick. From this all the joint velocities are calculated, but we only use the telescope velocity $\dot{q}_{4_{\text{ref}}}$.

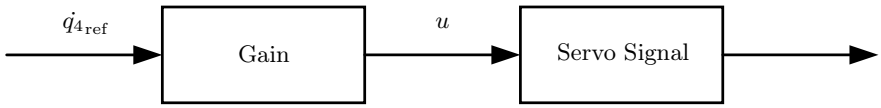

Figure 5.6: q_4 Velocity Open Loop Control

The velocity reference $\dot{q}_{4_{\text{ref}}}$ gets multiplied by a gain presented in Figure 5.6, before the control signal u gets scaled to actuate the servo motor for the telescope valve. Later in the tuning of the control system, we have the flexibility to experiment with some different gains, based on how active the telescope is for some given path. Here, a different gain can also be used based on the direction of $\dot{q}_4$.

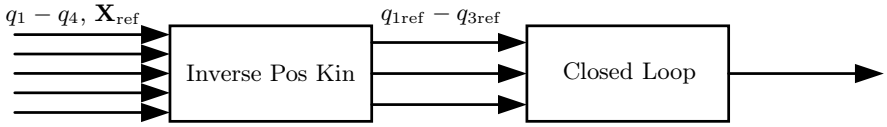

Figure 5.7: Inverse Position Kinematics

Now, the telescope is having some position, the 3 DOF inverse kinematics can be solved. This is the advantage of controlling q_4 in parallel, because at this time we control 3 coordinates in Cartesian space with 3 joints, meaning the system is no longer actuator redundant. Figure 5.7 shows a

simplified version og the inverse block. Based on all the joint positions $q_1 - q_4$, the desired joint position reference $q_{1\text{ref}} - q_{3\text{ref}}$ to satisfy the desired position $\mathbf{X}_{\text{ref}}$ can be found. These reference positions are used in three different closed loop controllers to make sure they stay as close as possible to their desired reference.

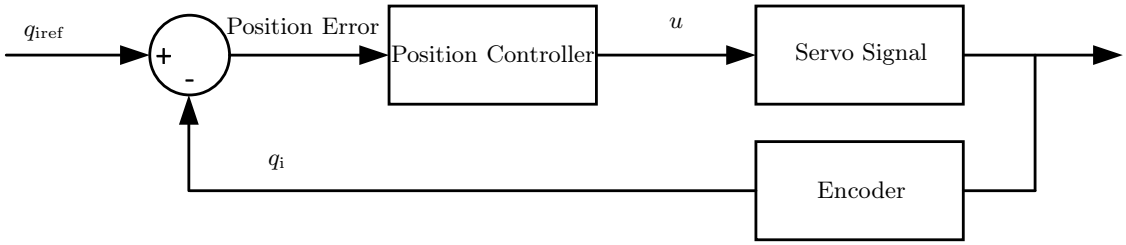

Figure 5.8: Joint Controller

Figure 5.8 shows the closed loop controller for one of the joints. There is 3 of them with different gains, servo signal scaling, and encoder value. The error between the reference position and the actual position is used together with a controller to actuate the servo motors for the three valves controlling $q_1 - q_3$. The position controller used for joint $q_1 - q_3$ is tested using P, PD, PI, PID controller and presented in the results chapter. The PID controller for joint q_1 can be represented in continuous time as:

$$u_1(t) = K_{P1}e_1(t) + K_{I1}\int_0^t e_1(\tau)d\tau + K_{D1}\frac{d}{dt}e_1(t) \tag{5.2}$$

The same equation goes for q_2 and q_3 as well, just changing all the indexes to 2 and 3. Different gains K_P, K_I, and K_D are used for the different test.

5.4 Heave Compensated Control

The heave compensated control is working the exact same way as the Cartesian controller. The only difference is that the desired Cartesian position needs to be translated from crane coordinate {s} to frame coordinate {f} before disturbance, and translated back to crane frame {s} when the disturbance is there and the tipping angle $\theta_{\text{tip}} \leq 0$. This is explained and simulated in the Multibody Dynamics chapter. It works both when the end effector is not changing position, and when the Cartesian control is active with a moving end effector.

Chapter 6

Software

This chapter presents the software of the demo model. First the architecture on how the different controllers communicate, the software for the EtherCAT slaves, and the IPC program itself. The software for all controllers is uploaded to the UIADemoCrane public repository on github [26].

For the slave software in the Arduino Controllers, Arduino IDE is used. Because Arduino is open source and have a wide range of users, there are a lot of open source libraries available in the IDE. Many pre made libraries are used in the following software. All the libraries are not cited directly, but they are mentioned in the UML class diagram, with the name of the library. Therefore, they can be found directly in the IDE or online [5]. Example of libraries used, Wire, AS5600, Adafruit_PWMServoDriver etc. Some processing are also done within the Arduino software.

6.1 Architecture

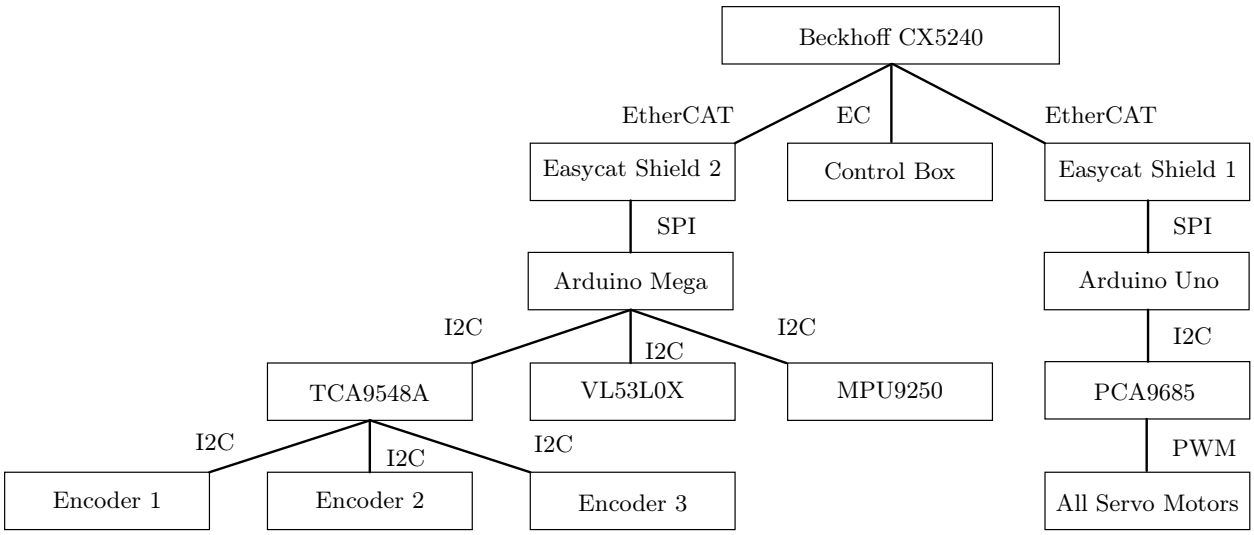

Figure 6.1: Software Architecture

The software is built up with the Beckhoff IPC as the top level master. Here all the logic on controlling the crane, the kinematic calculations for Cartesian control, integrator, controllers and much more. The IPC is communicating with the slaves through EtherCAT. The slaves are the control box, and the two Easy Cat shields. The control box is a extended part of the top level IPC, having a remote digital input and output, as well as analog input inside the box. This means all the logic regarding the control box, is written in the top level IPC program as well.

In comparison, the two EasyCat shields are sending and receiving some variables, such as servo value or encoder angles to and from the IPC. These values needs to be sent through different communication protocols, and some computation needs to be done inside the "slaves". All the different communication protocols between the components are presented in Figure 6.1. The sensors communicate with the Arduino using Inter-Integrated Circuit (I2C), the Arduino with the EasyCAT shield with Serial Peripheral Interface (SPI), and the Arduino communicating with the servo motors over Pulse Width Modulation(PWM).

6.2 Slave Software

6.2.1 EasyCAT Shield

The EasyCAT shield is made for Arduino. The task of the shield in this project is to send variables from the Arduino, through the EtherCAT signal to the master IPC, and send the variables from the IPC to the Arduino. From busano.net [2], all the necessary instructions and software is found. For this project, there are two different EasyCAT shields. One for all the inputs, and one for all the outputs. The variable name linked to each other are presented in Table 6.1.

I/O IPC	Arduino	EasyCatVariable	IPC Variable	Type	Unit
I	radA	encoder1Rad	fEncoder1Rad	float	[rad]
I	radB	encoder3Rad	fEncoder2Rad	float	[rad]
I	radC	encoder3Rad	fEncoder3Rad	float	[rad]
I	Telescope-Distance_mm	Telescope-Distance_mm	fTelescope-Distance_mm	float	[mm]
I	xRot	xRotDeg	fxRotDeg	float	[deg]
I	yRot	yRotDeg	fyRotDeg	float	[deg]
I	mpuCalibrated	mpuCalibrated	mpuCalibrated	uint8_t	[-]
O	ServoPulseLength1	ServoPulseLength1	fServoPulseLength1	int	[μs]
O	ServoPulseLength2	ServoPulseLength2	fServoPulseLength2	int	[μs]
O	ServoPulseLength3	ServoPulseLength3	fServoPulseLength3	int	[μs]
O	ServoPulseLength4	ServoPulseLength4	fServoPulseLength4	int	[μs]
O	ServoPulseLength5	ServoPulseLength5	fServoPulseLength5	int	[μs]
O	ServoPulseLength6	ServoPulseLength6	fServoPulseLength6	int	[μs]

Table 6.1: Variables Linked

Figure 6.2: Easy Configurator

There are some important steps on how to configure the EasyCAT board, implement it as a slave in the IPC, and add the header file to the arduino project running on the Arduino. Firstly, the variables necessary to pass, their type, and if its output or input, needs to be created in the Easy Configurator. Figure 6.2 shows the screen capture of the Easy Configure app for the inputs on the left side, and the outputs on the right side. When the variables are defined, the EasyCAT shield needs to be configured by writing EEPROM, with the EasyCAT shield connected to the computer through the Ethernet cable. Then the header file is copied by pressing Copy .h file, and placed in the same folder as the program running on the Arduino. Then, to find the slave inside the IPC program, the .xml file needs to be placed in the right folder. For TwinCAT 3 on windows, the .xml file is placed in the EtherCAT folder (`C:/TwinCAT/3.1/Config/Io/EtherCAT`). Now, the slave can be found in the IPC program, when connected to the IPC with EtherCAT.

6.2.2 Outputs

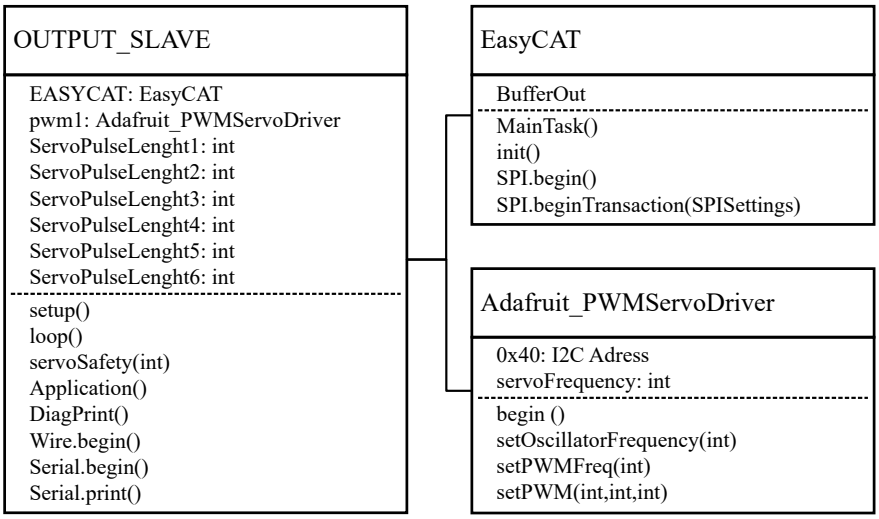

Figure 6.3: Arduino Output

The task of the output slave Arduino is to receive the pulse length of the servo PWM from the IPC, and write it to the different sensors. The UML Class diagram of the Arduino software is presented in 6.3. The EasyCAT library is downloaded from busano.net [2]. Also examples on how it is implemented in the Arduino program is provided. The slave Arduino script is attached in appendix B.2.

Figure 6.4 shows the flowchart for the Arduino output slave. The serial print of the values is for diagnosis purpose. If there are some doubt about the slave working properly, the serial USB cable can be connected to a laptop and check the values are printing.

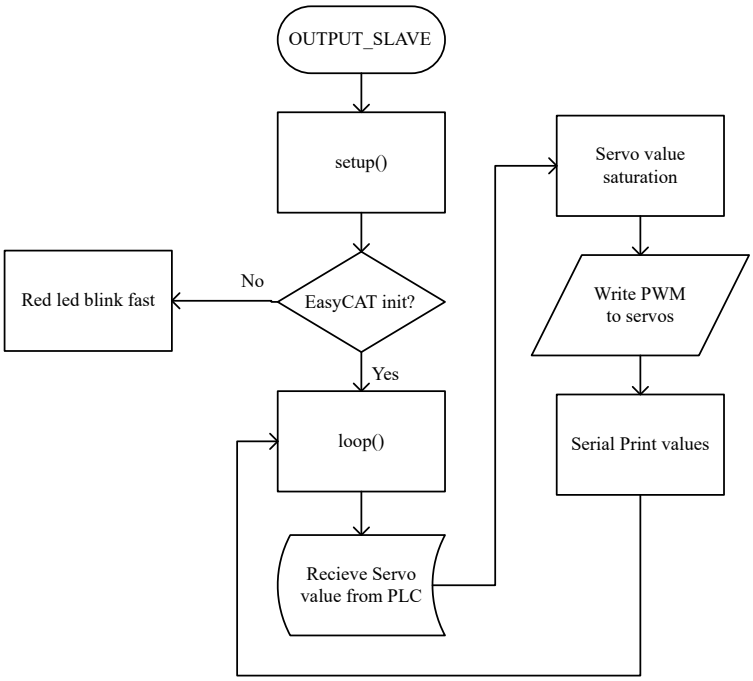

Figure 6.4: Flowchart Arduino Output

6.2.3 Inputs

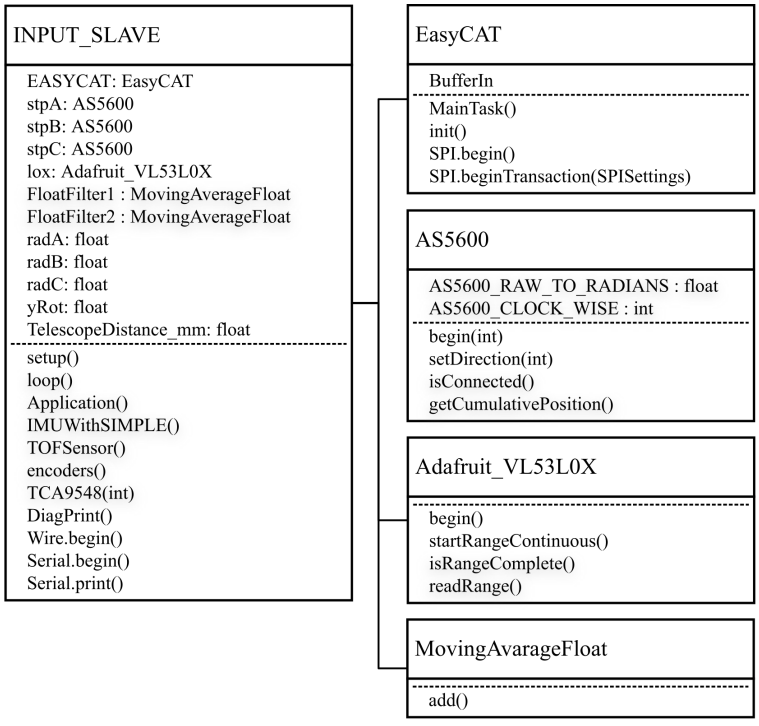

Figure 6.5: Arduino Input

The task of the input slave Arduino is read all the sensors, filter and process, and send to the IPC. The UML Class diagram of the Arduino software is presented in Figure 6.5. The library AS5600 is for the magnetic encoders, Adafruit_VL45L0X is for the TOF sensor, and MovingAvarageFloat is for the average filter.

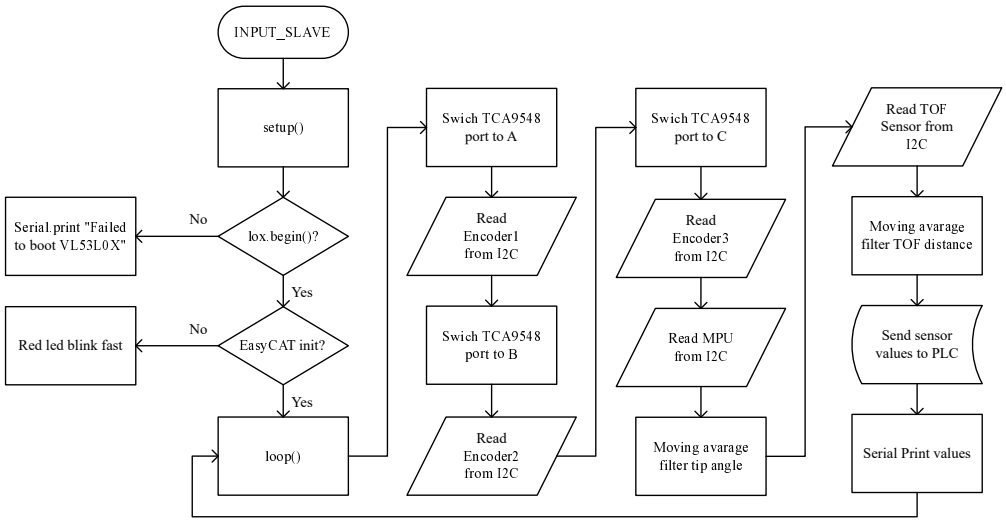

Figure 6.6: Flowchart Arduino Input

Figure 6.6 shows the flowchart of the input slave. Note the I2C switch TCA9548 switching port before reading encoders. This is because all the encoders have the same address. Also, the moving average filter is processing the signal from the TOF sensor as well as the angle signal from the MPU. The moving avarage filter is using the avarage value of n samples, meaning the signal have less spikes, but this also introduce some delay. The slave Arduino script is attached in appendix B.1.

6.3 IPC Software Interface

TwinCAT 3 is the software running on the Beckhoff IPC [12]. For the programming of the IPC, the software TwinCAT XAE is used. This section is explaining how to connect the IPC to the slaves, as well as how the top level IPC program is build up and working.

6.3.1 Program Tree and Hardware

Figure 6.7: Add Slaves TwinCAT

Figure 6.7 shows the project tree on the left side. The folders are containing different functions and function blocks that is explained in more detail later. The main programs are placed in the POUs folder. The Human Machine Interface (HMI) is also in tho POUs folder.

Figure 6.7 shows the hardware tree on the right side. There is feature in TwinCAT that makes it possible to scan for devices to find hardware. This worked well for the Beckhoff hardware, like the top level IPC and the control box. For the custom slaves, they need to be added manually from the .xml file from Easy Configurator, placed in the right folder. This is done by right clicking somewhere under the top level IPC, and click add new item. From there, scroll to AB&T and add the slave configuration.

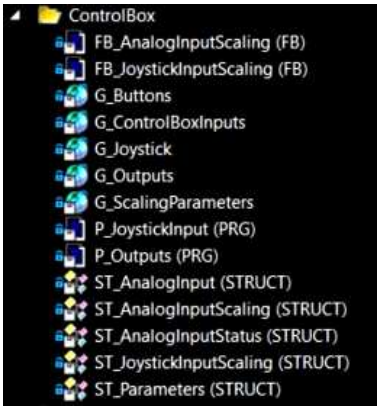

Figure 6.8: Control Box Software

Figure 6.8 shows the folder of the control box software. This software builds on software presented by Daniel Hagen in course MAS418 at UiA [16]. The program and function blocks are handling the raw data coming from the analog joysticks and normalize them from -1 to 1, and includes some dead band. Because the joysticks are modified to have one more button for each joystick, the software is also modified to include this.

6.3.2 HMI

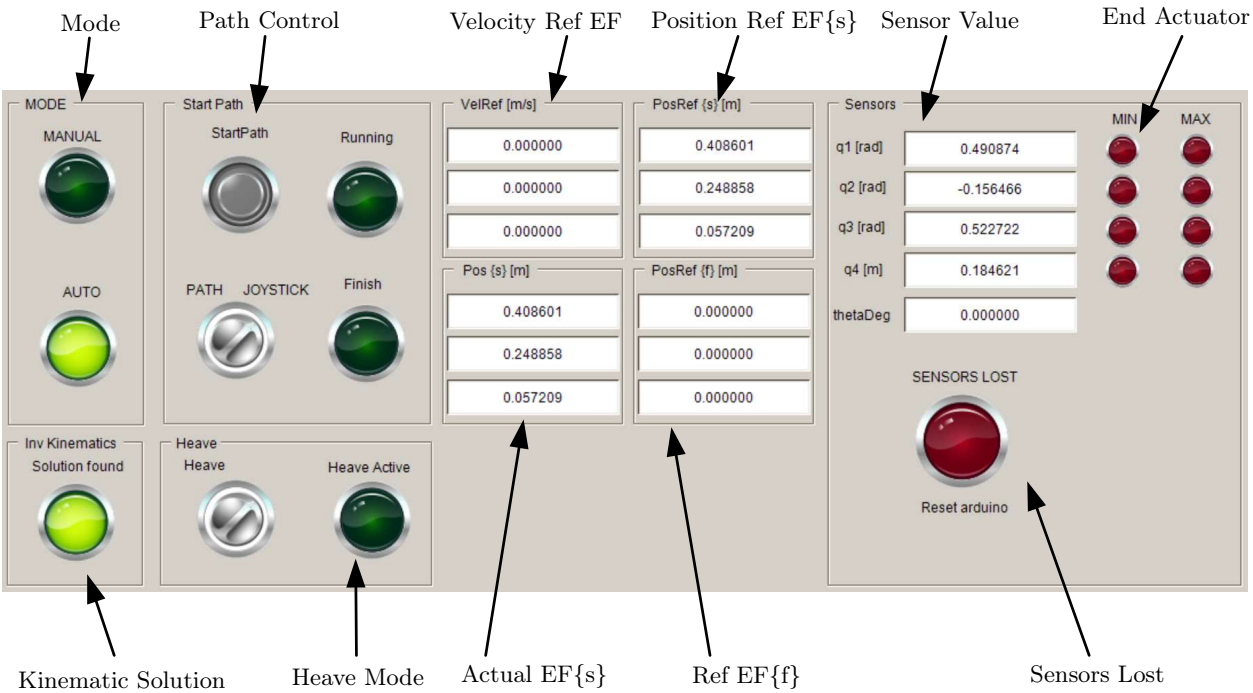

Figure 6.9: HMI

Figure 6.9 is showing the HMI. The HMI is a necessary tool for some of the features of the crane. The mode button on the control box can switch between manual and Cartesian mode. However, when the button is in Cartesian (auto) mode, it is possible to choose between Cartesian control with the joystick, or using a pre made path. The pre-made path is mainly ment for tuning the control system, but can also be used on demonstrations. The heave compensation can also be switched on and off using the HMI.

The sensor values are displayed at all time. The warning lamps are also indicating if the actuators are close to end position. When using Cartesian controller, the actuators stops when they are close to the end position. In manual mode however, the lamps are lit, but the crane is still operative. This is because the crane should not be compromised in manual mode, in case there are some problems with the sensor. If the sensor communication is lost, the big red LED is turned on, and auto mode is unavailable. If the crane is running in auto mode, the mode is automatically switching to idle.

6.4 IPC Program

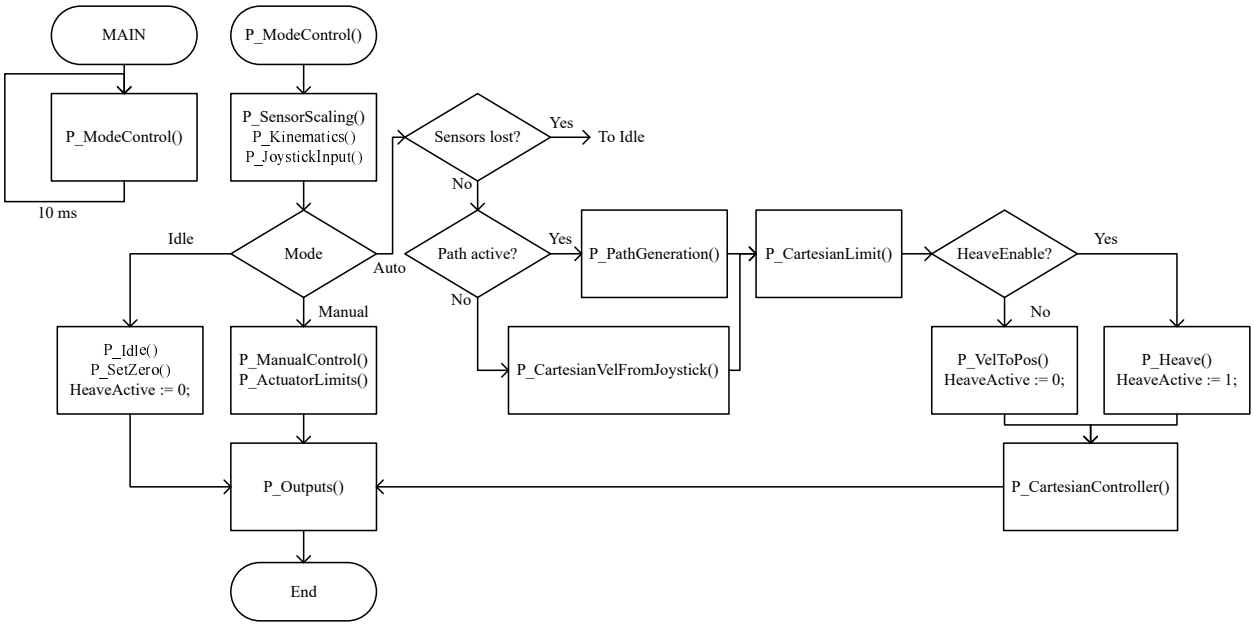

Figure 6.10: IPC Top Level Flowchart

The top level flowchart for the IPC program is presented in Figure 6.10. The only program above P_ModeControl is MAIN. MAIN runs with a cycle time of 10 [ms]. The Class diagram of the IPC software is exported from TwinCAT and added in appendix C.2. The class diagram is not including the Matlab generated functions and function blocks. The flow of the IPC program will be explained further with the sub-charts including the programs presented in the top level chart.

The three main modes are Manual mode, Idle and Auto. Auto mode is Cartesian controller. The different modes will be presented separately. Firstly, the flowcharts that affects all modes will be explained.

6.4.1 All Modes

Figure 6.11 shows the flow of sub programs P_SensorScaling(), P_Kinematics(), P_JoystickInput() and P_Outputs(). These programs will be called every cycle time as presented in the top level flowchart.

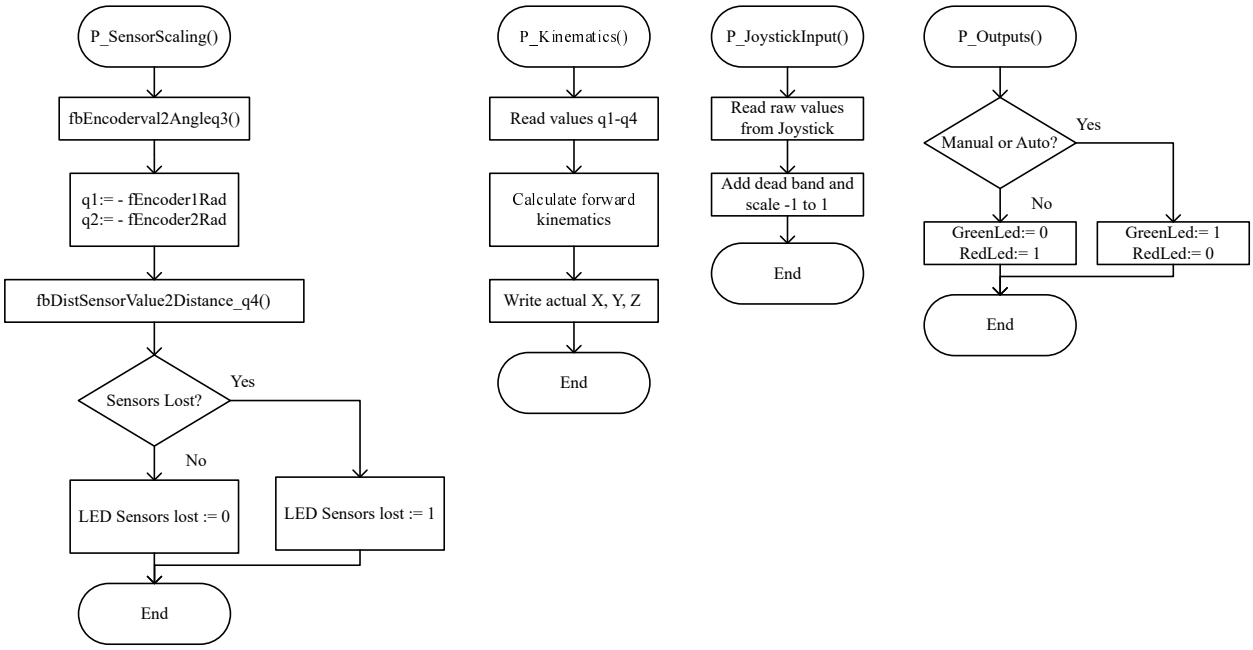

Figure 6.11: Sub Charts For All Modes

P_SensorScaling() is scaling the sensor values to fit the kinematics of the crane. The function block fbEncoder2Angleq3() is the directly inserted from the PLC code generation function in Matlab. The Matlab function is presented in appendix B.4. This is converting the sensor value from encoder 3 to the angle q3. fbDistSensorValue2Distance_q4() scales the sensor value from the telescope to range from 0 - 0.2 [m].

P_Kinematics() is also using a converted Matlab function presented in appendix B.5 to calculate the X, Y, and Z value of the end effector.

6.4.2 Manual

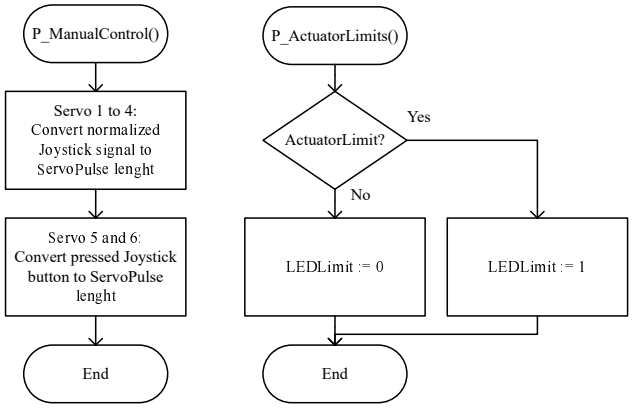

Figure 6.12: Sub Charts Manual Mode

Figure 6.12 show the flow of P_ManualControl() and P_ActuatorLimits(). For servo 1 - 4 the joystick signal is proportionally scaled for the servo motor. However, for servo 5 and 6, the servo is set to a fixed value based og the buttons. P_ActuatorLimits is used to light the warning LED in the HMI, and restrict the actuators in Auto mode.

6.4.3 Idle

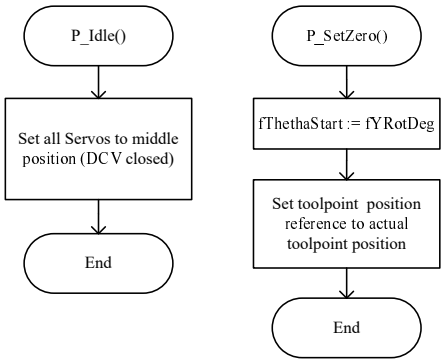

Figure 6.13: Sub Charts Idle

Figure 6.13 show the flow of P_Idle() and P_SetZero(). Idle mode is keeping all actuators still by setting the servo motors in zero position. P_SetZero() is setting the start angle of the tilt handle to the current angle, as well as the end effector reference to actual end effector position.

6.4.4 Auto

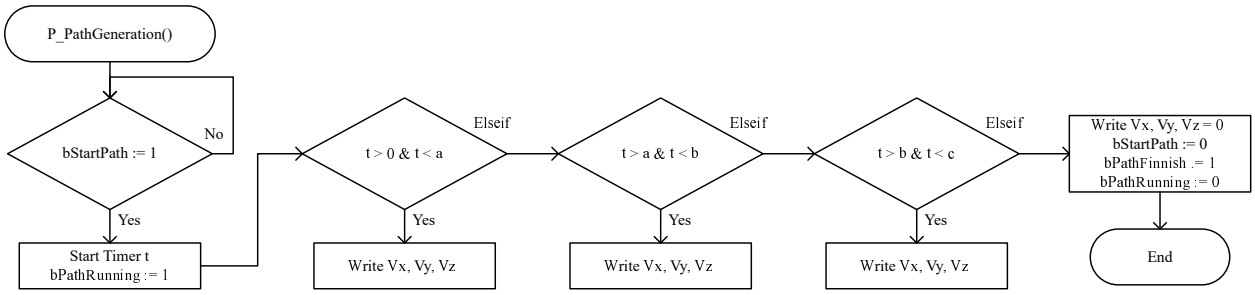

Figure 6.14: Sub Chart Path Generator

Figure 6.14 show the flow of P_PathGeneration(). This is a very simple path generator, generating a linear velocity reference for a certain amount of time before going to next time slot. Some different path generators are used through out the project. The one in the chart is a illustration showing an example.

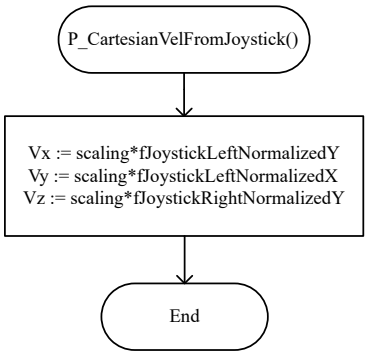

Figure 6.15: Sub Chart Joystick to Cartesian Velocity

Figure 6.15 shows the conversion of the normalized joystick signal to Cartesian velocity.

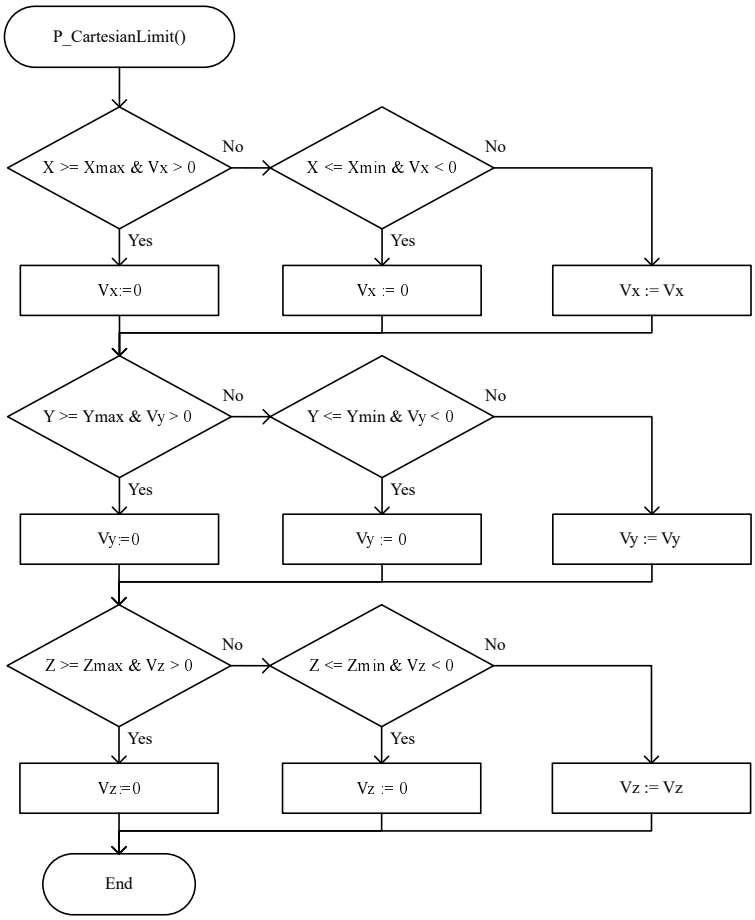

Figure 6.16: Sub Chart Cartesian Limit

Figure 6.16 shows the flowchart of the Cartesian limit. This is restricting the end effector from running outside the desired workspace.

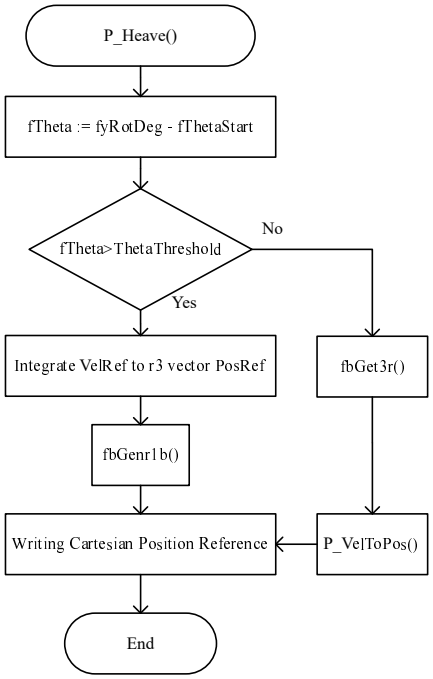

Figure 6.17: Sub Chart Heave

Figure 6.17 shows the flow of P_Heave(). If the tilt is tilted, the frame end effector is controlled in the frame coordinate {f}, else the crane is controlled in crane coordinate {s}.

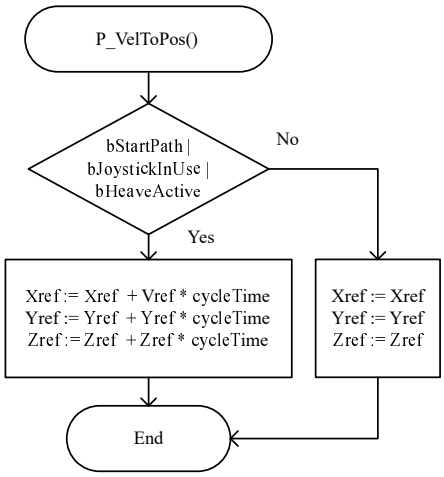

Figure 6.18: Sub Chart Cartesian Velocity Integration

Figure 6.18 shows the flow of P_VelToPos(). This is setting the end effector position reference based on the end effector velocity by integration.

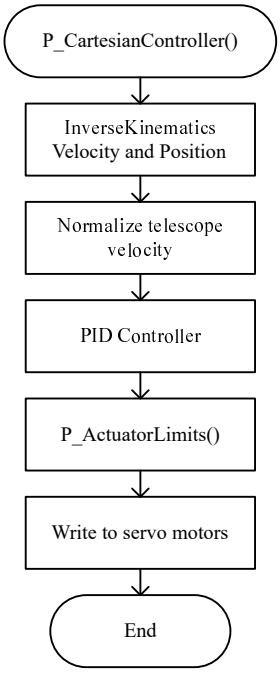

Figure 6.19: Sub Chart Cartesian Controller

Figure 6.19 shows the flow of P_CartesianController(). First the inverse kinematics are calculated using the functions generated from the Matlab function presented in appendix B.8. Then the telescope velocity is normalized, based on the velocity signal coming from the velocity kinematics and the maximum possible telescope velocity. From there, three different PID controllers control the joint position of q_1, q_2, and q_3. P_ActuatorLimits() check if the actuators are close to end position, and prevents movement towards end if they are. Then, the servo motors are actuated.

Chapter 7

Outcome and Results

Because the objective of this thesis is to design, build, program, and control a demo model, the first section of the results have pictures of the demo model. The next section will show some results regarding the manual control, before the last section will present several different controllers for the Cartesian controller as well as the heave compensated control.

7.1 Demo Model

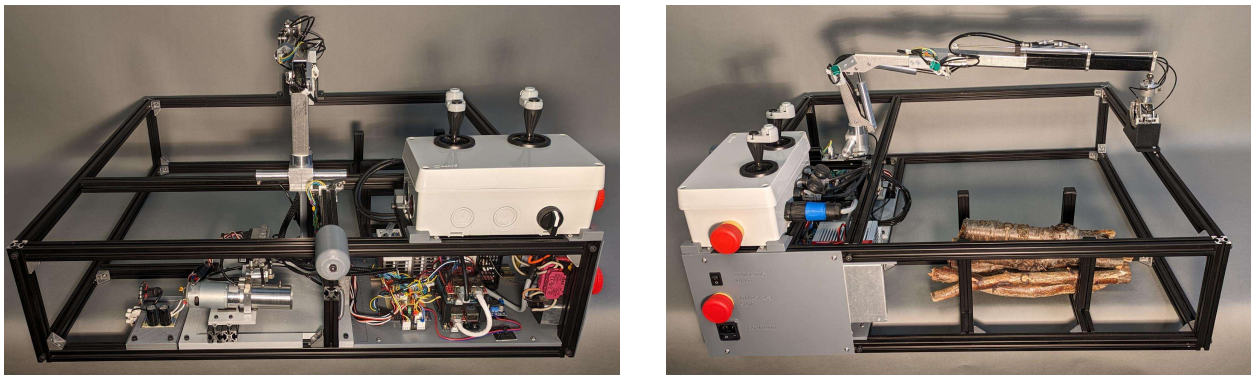

Figure 7.1: Demo Model Side View

Figure 7.1 shows the demo model with the crane at initial position. The crane is parked in this position to reset the encoders. Meaning all angles $q_1 - q_3 = 0$ with this crane configuration. There are small timber logs are added in the tray inside the workspace.

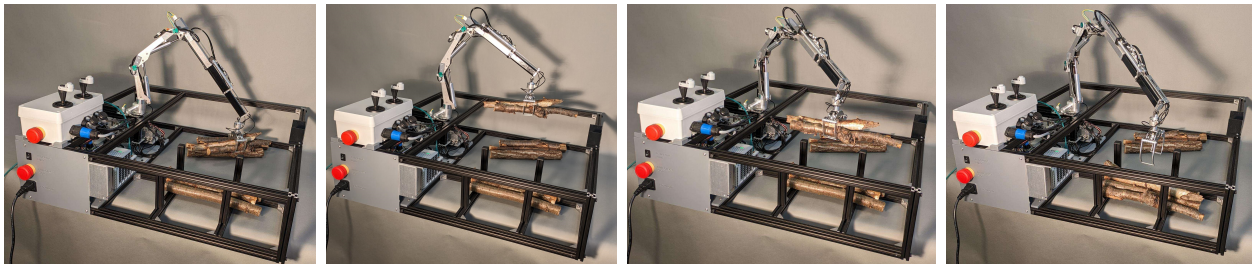

Figure 7.2: Place Logs In The Tray

The demo model is tested several hours, lifting dummy timber logs as presented in Figure 7.2. The four pictures shows one lifting sequence for the timber to be stacked in the tray.

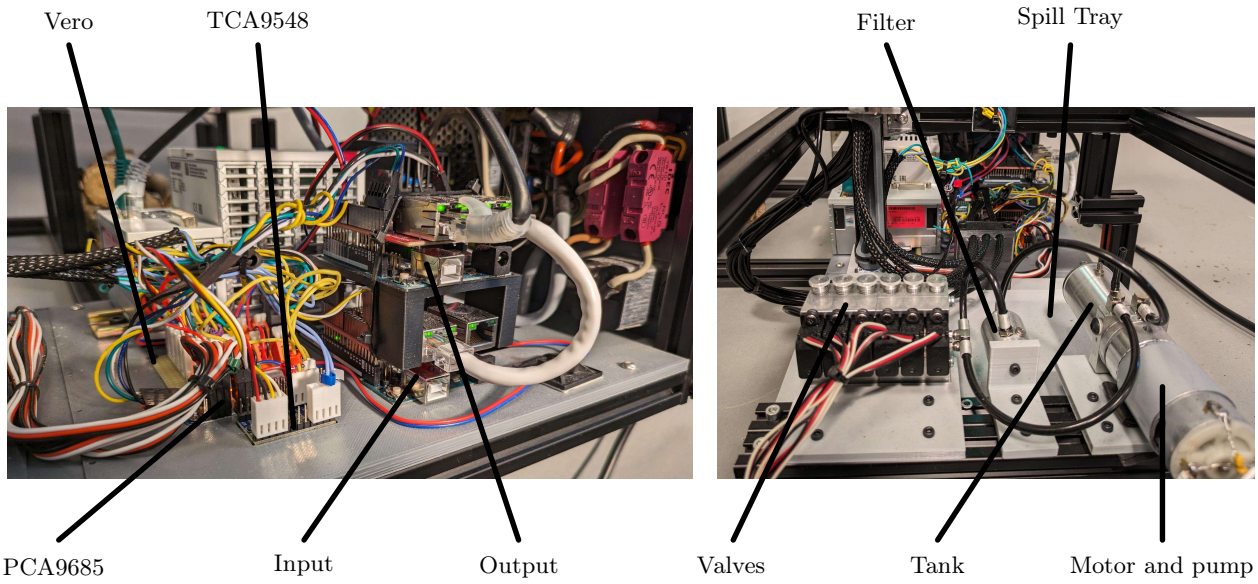

Figure 7.3: Electric and Hydraulic Components

Figure 7.3 shows some of the electric components on the left and the hydraulic components on the right. Note the input Arduino is the bottom board in the stack.

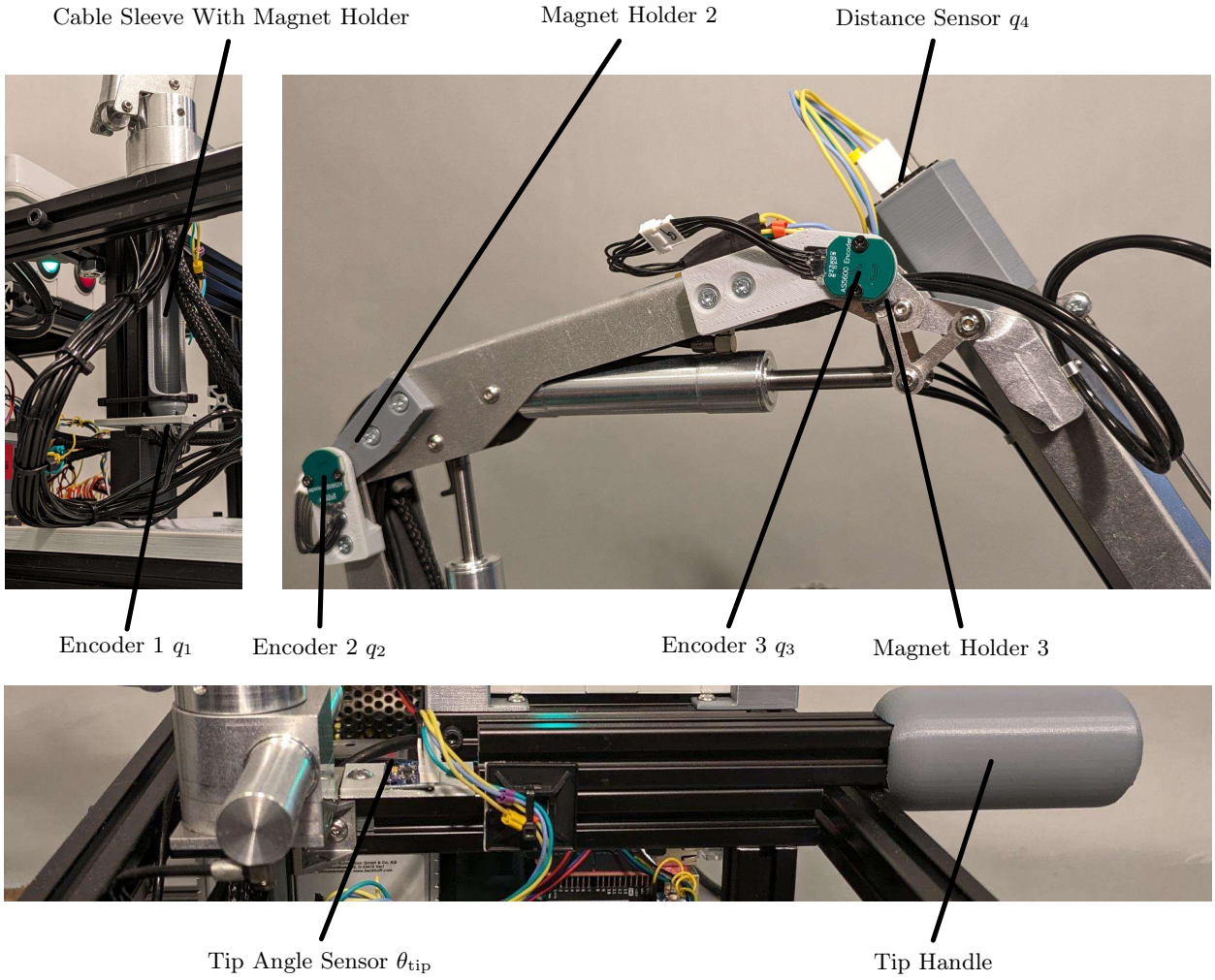

Figure 7.4: Sensor Attachment

The demo model was successfully designed and build using small sensors, and discrete wiring as presented in Figure 7.4. All the sensor cables fitted inside the column, and was attached properly. The crimped cables with connector headers on the Veroboard created a clean and stable result for

the electronic connections.

Figure 7.5: Clamping

Because one of the most important tasks of the demo model is to pick up logs, this is presented in Figure 7.5.

7.2 Manual Controller

Because there is little to none information about the DCV characteristics, some test are done actuating one servo motor at the time with a sinusoidal wave, measuring the actuator velocity for the respective joint. The sinusoidal wave is ranging from 420 to 280 [μs] pulse length, corresponding to approximately 60 degrees variation for the servo motor. Even though the dynamics such as force, moments and moment of inertia are neglected, it is interesting to see the difference in actuator velocity with different crane configurations.

Test for q_1 - q_3 are presented in Figure 7.6, 7.7 and 7.8. Here, one actuator test is done with the telescope fully retracted and one fully extracted.

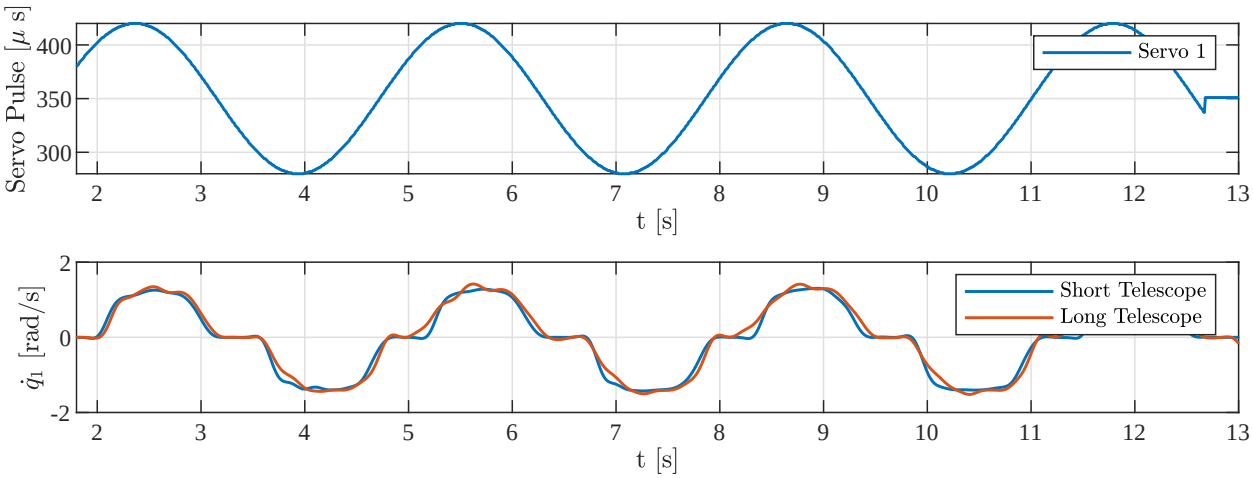

Figure 7.6: q_1 Servo Position vs Joint Velocity

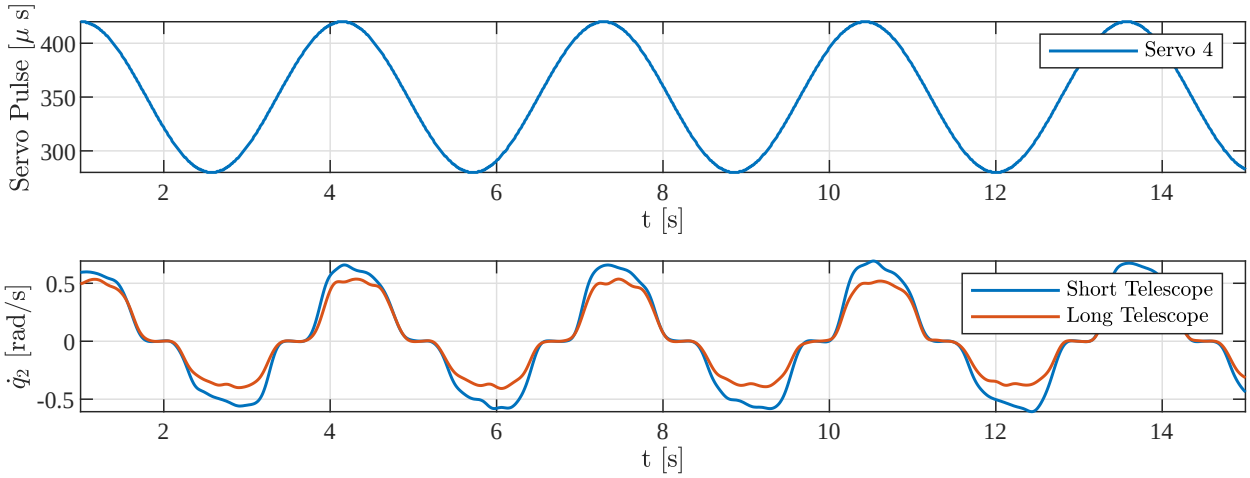

Figure 7.7: q_2 Servo Position vs Joint Velocity

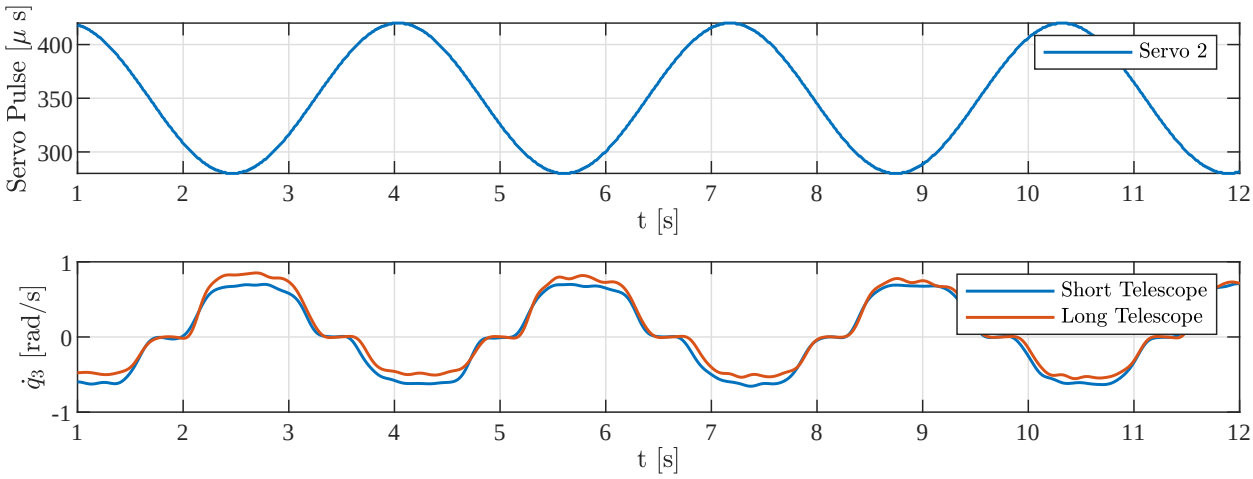

Figure 7.8: q_3 Servo Position vs Joint Velocity

Interesting observations for q_1 - q_3: The velocity curve is not proportional to the the sinus wave, and dependent of the telescope length q_4. Maximum velocity is also dependent on the direction.

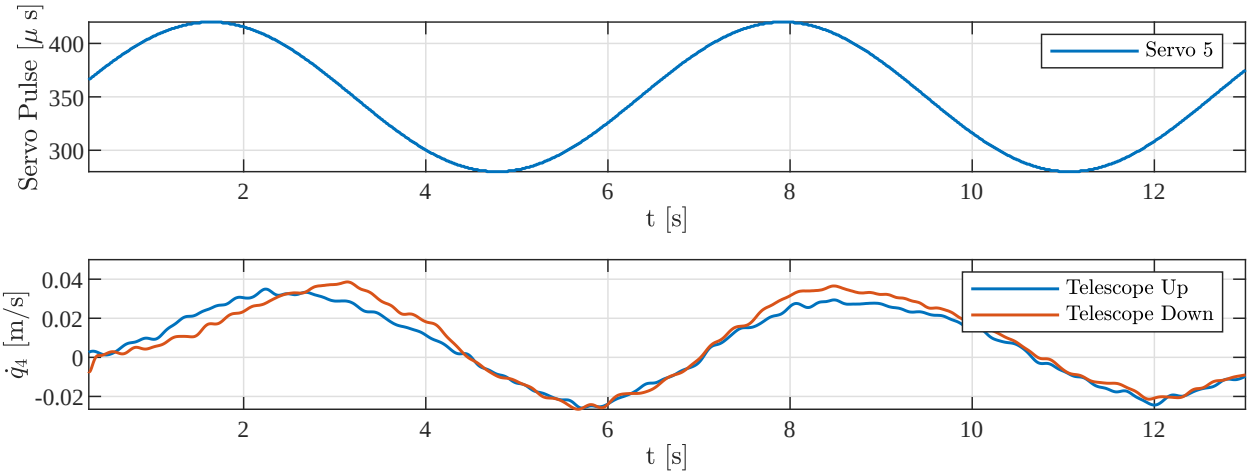

Figure 7.9: q_4 Servo Position vs Joint Velocity

Test for q_4 is done with the telescope pointing upwards, and one with the telescope pointing downwards. The result is presented in Figure 7.9. Small difference between upwards and downwards, however large difference between $\dot{q}_4 > 0$ vs $\dot{q}_4 < 0$.

7.3 Cartesian Controller

For the Cartesian controller results section, different configurations and controllers are tested. First, the telescope is locked, and the crane is following a Cartesian position reference using three P-controllers, one for each joint. Then the telescope is activated, and the results following the same reference signal is measured. Now, P, PD, PI, and PID controllers are tested for joint $q_1 - q_3$.

The results are presented in different plots. The first plot is Cartesian reference position X_{ref}, Y_{ref}, Z_{ref}, is plotted against the actual end effector position X, Y, and Z. The difference between these two is the error e. For example $e_X = X_{\text{ref}} - X$.

Based on the inverse kinematics, the joint position reference $q_{1\text{ref}}$, $q_{2\text{ref}}$, and $q_{3\text{ref}}$ are found and plotted against the actual joint position q_1, q_2, and q_3. Here the error e is calculated and plotted as well. For example: $e_1 = q_{1\text{ref}} - q_1$

The telescope position q_4 is presented as well. This is open loop controlled and therefore there is no reference signal, only actual position.

7.3.1 P - controller 3 DOF

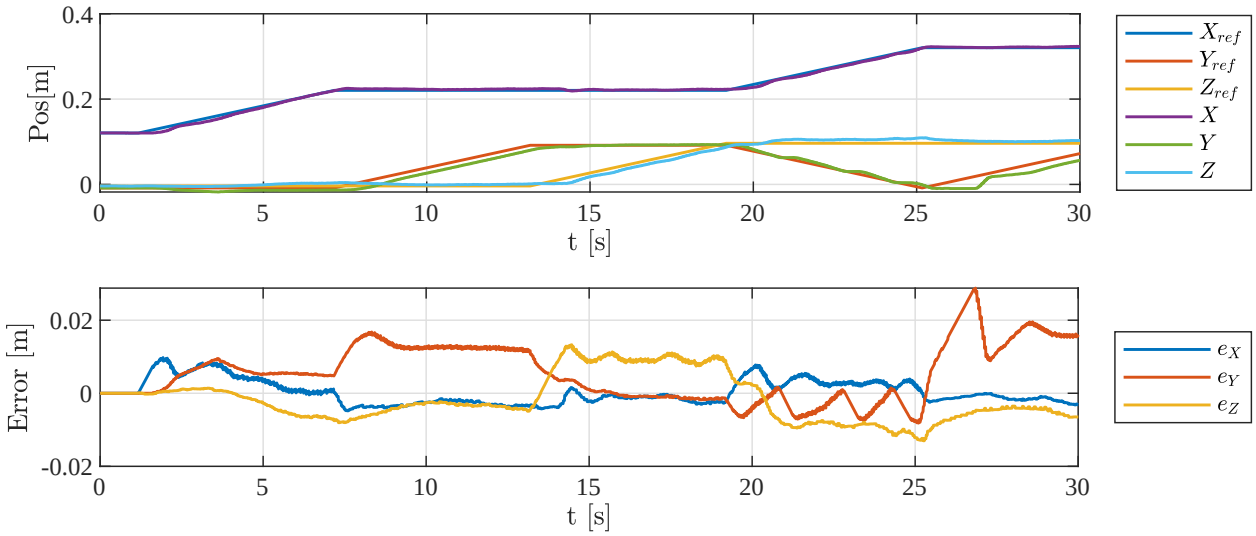

Figure 7.10: Cartesian Results Locked Telescope

Figure 7.10 shows the Cartesian results, and Figure 7.11 shows the joint results. This test is with the telescope q_4 locked at approximately 0.05 [m]. Note that the test was stopped at around $t = 30$ [s] because the crane did not reach the reference position with the telescope locked.

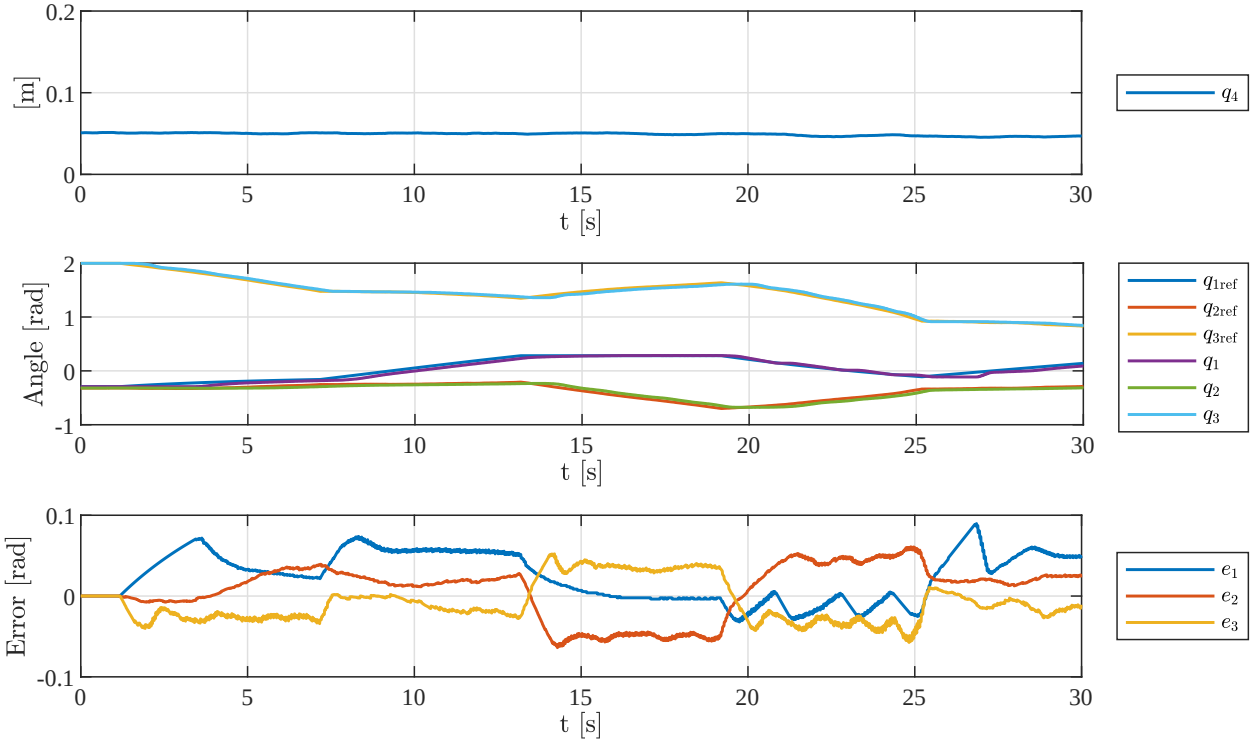

Figure 7.11: Joint Results Locked Telescope

The gains used for this test are presented in Table 7.1. Note that the telescope gain is 0 to lock q_4 to a constant position.

K_{P1}	K_{P2}	K_{P3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	0	0

Table 7.1: Gains P - Controller No Telescope

7.3.2 P - controller 1

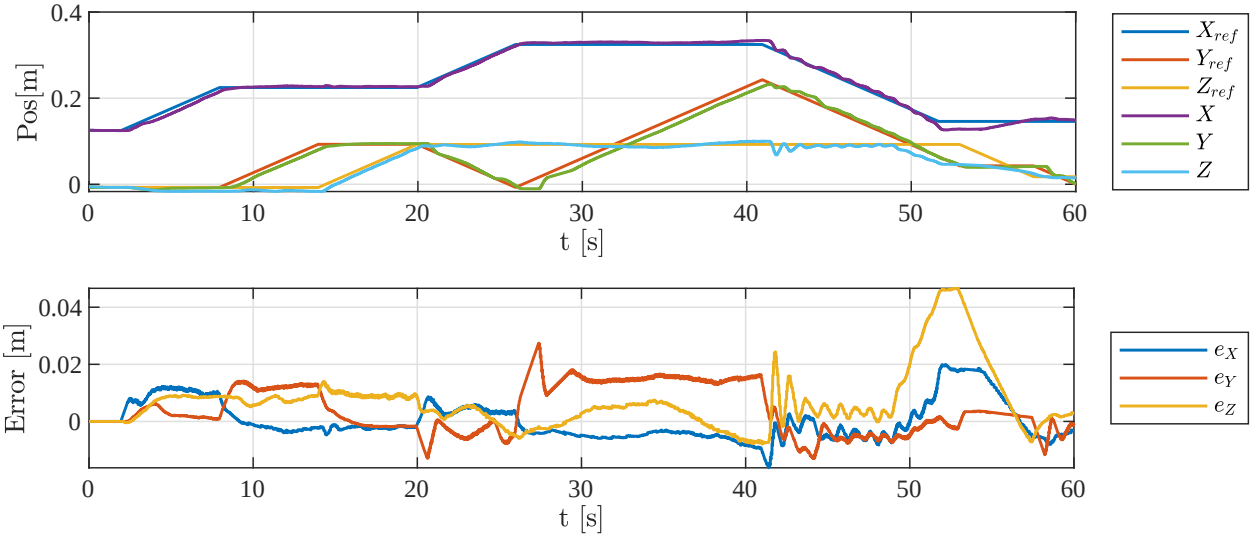

Figure 7.12: Cartesian Results P - Controller 1

Figure 7.12 shows the Cartesian results, and Figure 7.13 shows the joint results. The gains used are presented in Figure 7.2. Now the telescope is actuated. The first 30 seconds of the result is similar

to the locked q_4 above. This is showing that the joints $q_1 - q_3$ compensate for the open loop control of the telescope.

K_{P1}	K_{P2}	K_{P3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	0.05	8·0.05

Table 7.2: Gains P - Controller With Telescope

Around 50 [s] there is a sudden large error e_Z. This is because q_4 is not retracting enough, and tries to get q_2 to compensate for this error. This is causing q_2 to reach saturation at end stop, and the error occurs. A closer look at q_4 in Figure 7.13 shows that the telescope is almost fully extracted, even if the Cartesian position is close to the starting position.

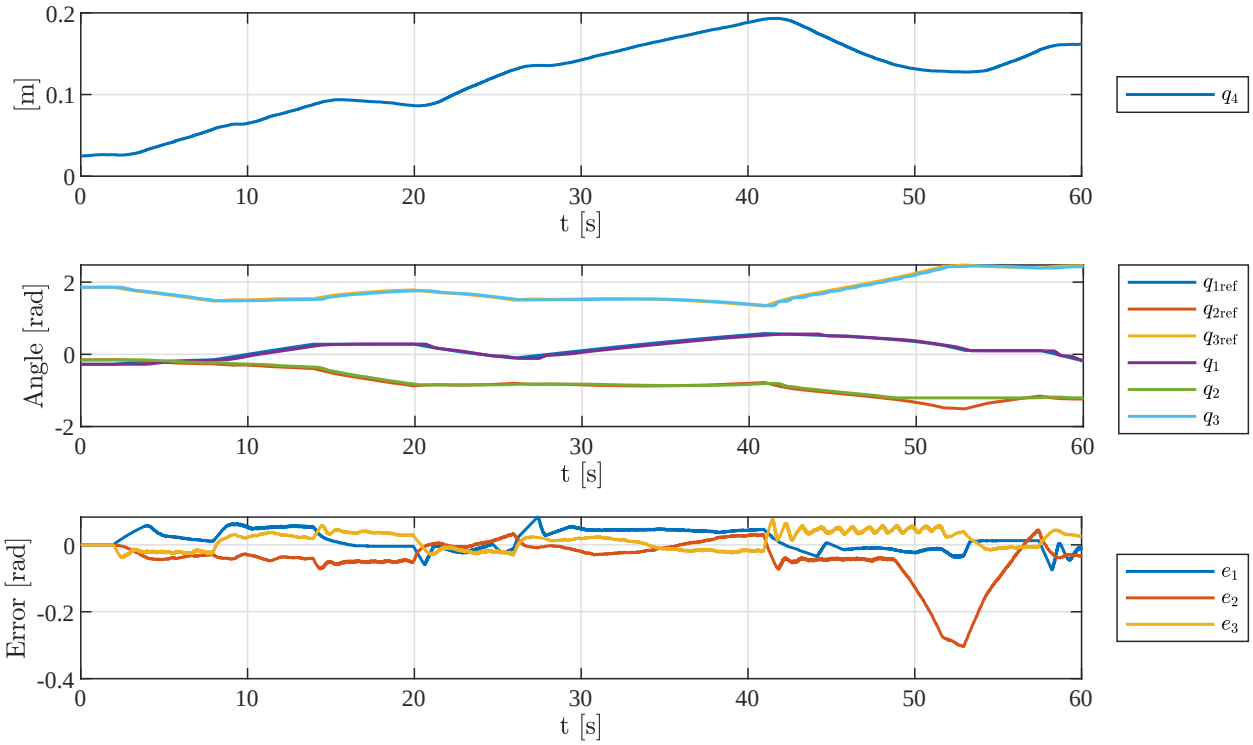

Figure 7.13: Joint Results P - Controller 1

7.3.3 P - controller 2

K_{P1}	K_{P2}	K_{P3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	0.05	10·0.05

Table 7.3: Gains P - Controller 2 With Telescope

Because the telescope was tending to extend in the last experiment, the gain for the return of the telescope $K_{\text{telescopeIN}}$ was increased even more. Now, the gain is 10 times stronger on the retraction than extraction.

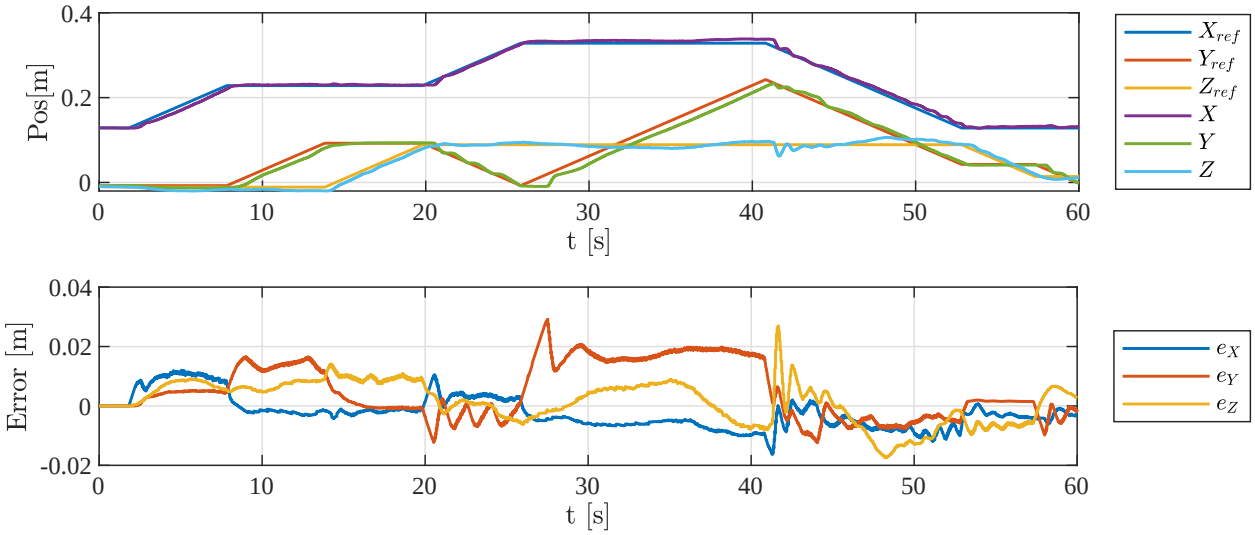

Figure 7.14: Cartesian Results P - Controller 2

Figure 7.14 shows the Cartesian results, and Figure 7.15 shows the joint results. The results looks similar to before, with a more active telescope joint. Now the telescope is stopping approximately at the same position as the start position, and no joints are reaching saturation.

The error is larger with a longer telescope, and there are oscillation especially at approximately 42 [s] when the end effector is staring to move in negative X, and Y direction.

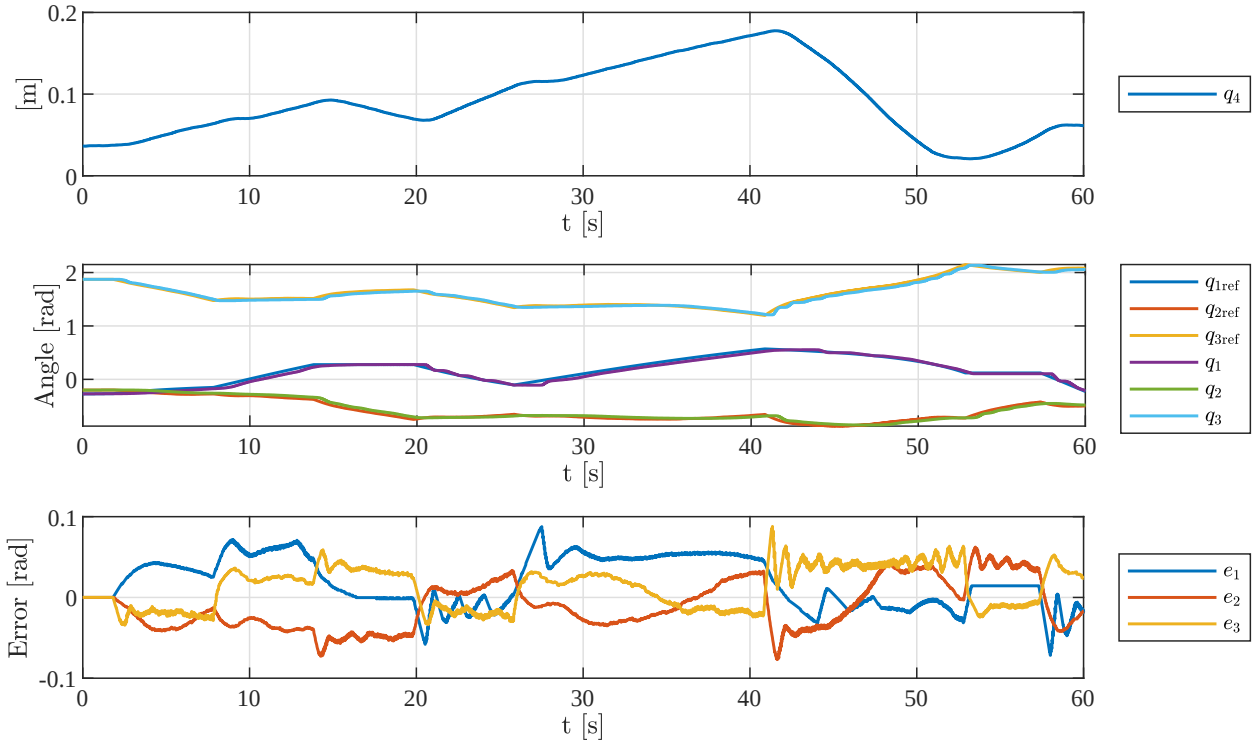

Figure 7.15: Joint Results P - Controller 2

7.3.4 PD Controller 1

K_{P1}	K_{P2}	K_{P3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	0.5	0.5	0.5	0.05	10·0.05

Table 7.4: Gains PD - Controller 1 With Telescope

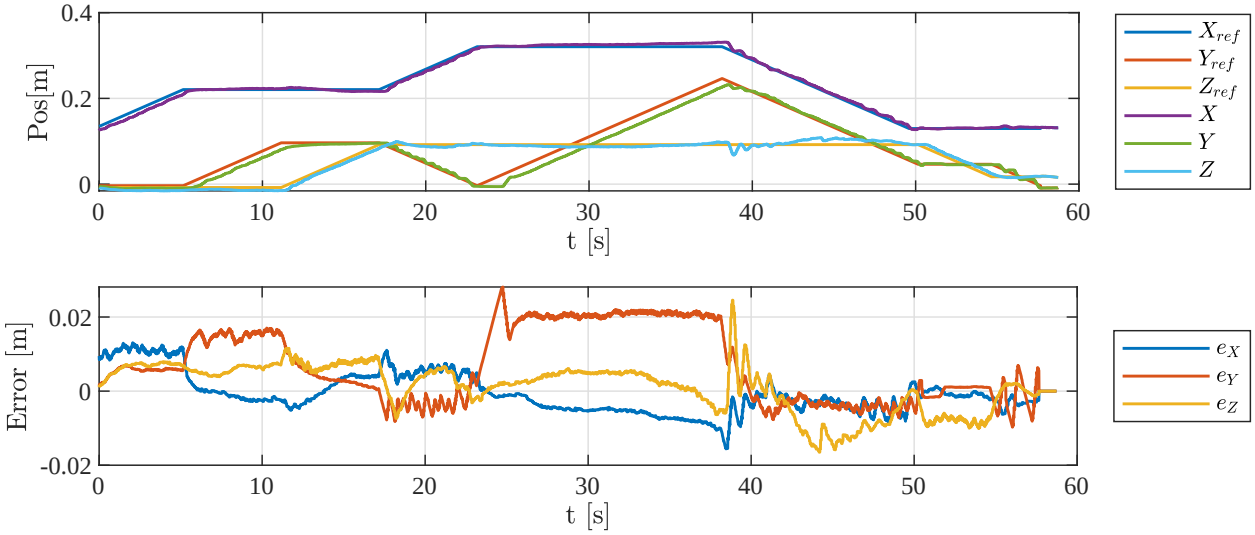

Figure 7.16: Cartesian Results PD - Controller 1

Trying to add a derivative part to decrease the settling time. The gains for the PD controller are now presented in Table 7.4. This reduced the amplitude of the osculations, but increased the frequency as presented in Figure 7.16 and Figure 7.17.

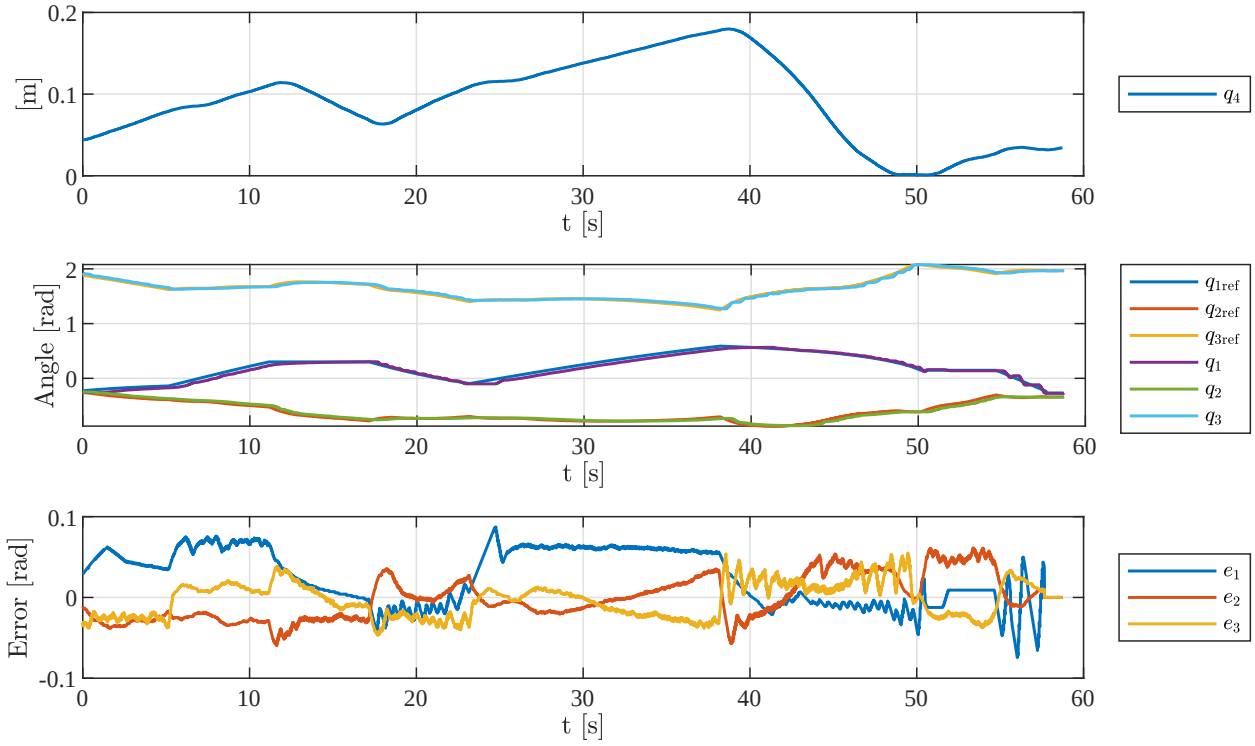

Figure 7.17: Joint Results PD - Controller 1

7.3.5 PD Controller 2

K_{P1}	K_{P2}	K_{P3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	0.1	0.1	0.1	0.05	10·0.05

Table 7.5: Gains PD - Controller 2 With Telescope

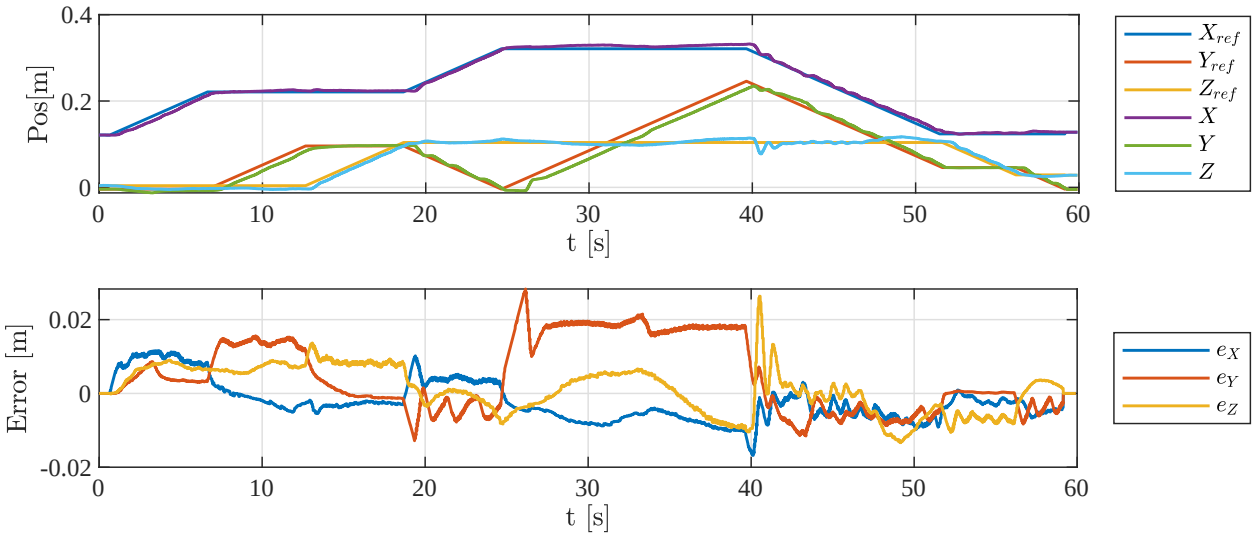

Figure 7.18: Cartesian Results PD - Controller 2

PD Controller 2 reduces the derivative gain, and shows less frequency and higher amplitude for the oscillations as expected.

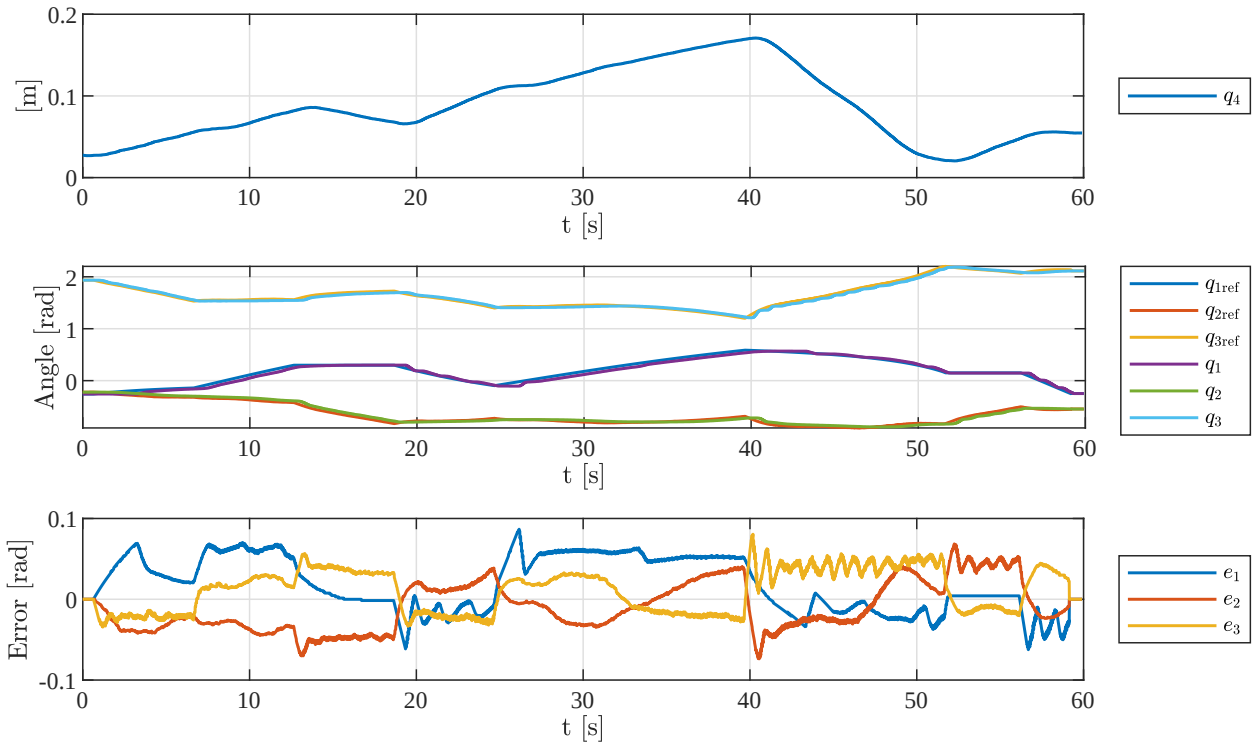

Figure 7.19: Joint Results PD - Controller 2

7.3.6 PID Controller 1

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	9	1	1	1	0.1	0.1	0.1	0.05	10·0.05

Table 7.6: Gains PID - Controller 1 With Telescope

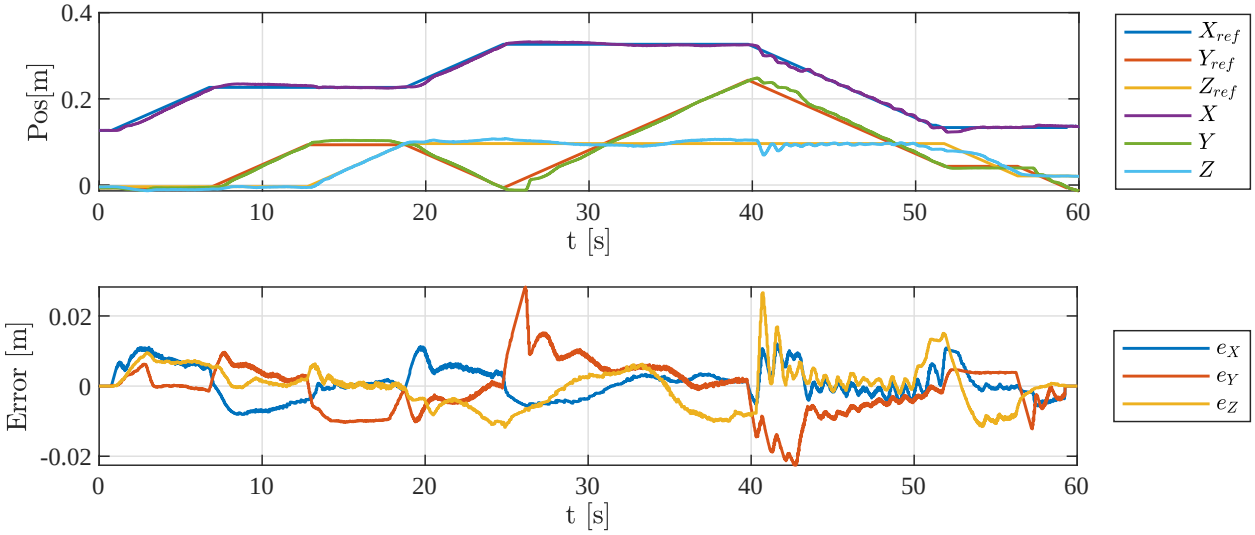

Figure 7.20: Cartesian Results PID - Controller 1

Adding the integral terms for each joints shows reduction in steady state error.

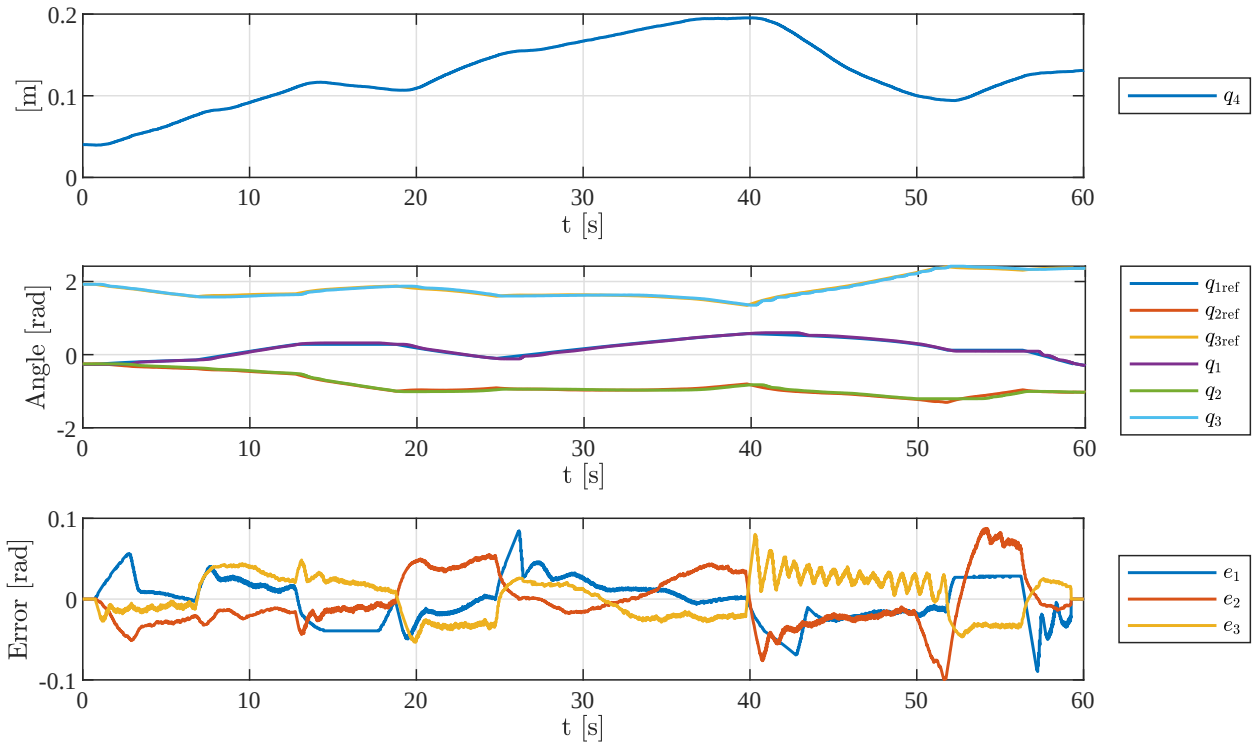

Figure 7.21: Joint Results PID - Controller 1

7.3.7 PID Controller 2

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	6	1	1	1	0.1	0.1	0.2	0.05	10·0.05

Table 7.7: Gains PID - Controller 2 With Telescope

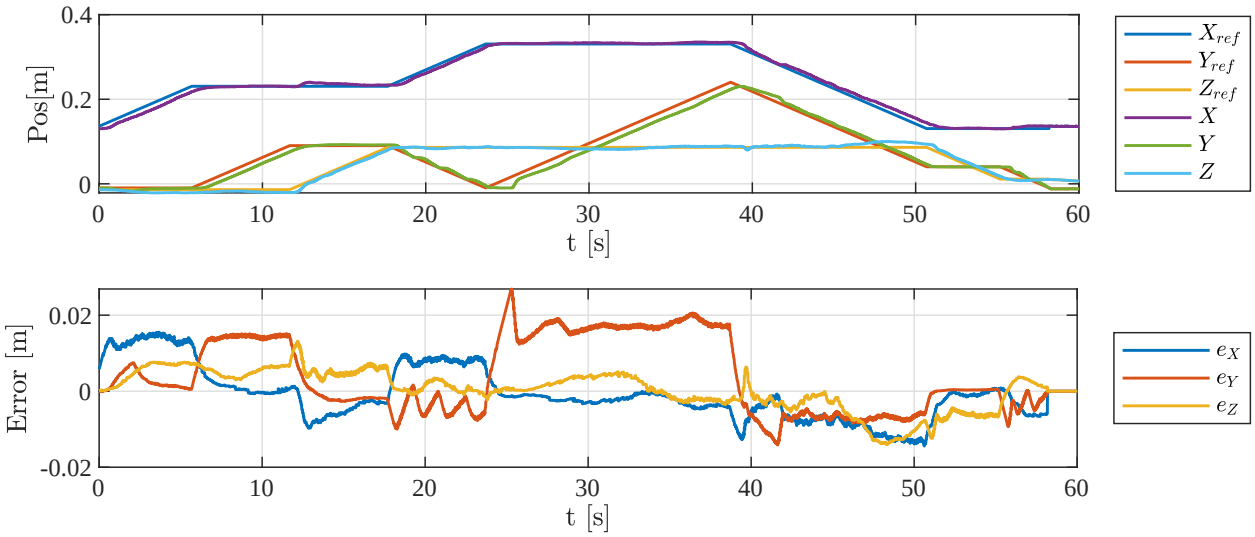

Figure 7.22: Cartesian Results PID - Controller 2

Because of oscillation on q_3 the gain K_{P3} is reduced and K_{D3} is increased. Some more error is introduced, but less oscillations.

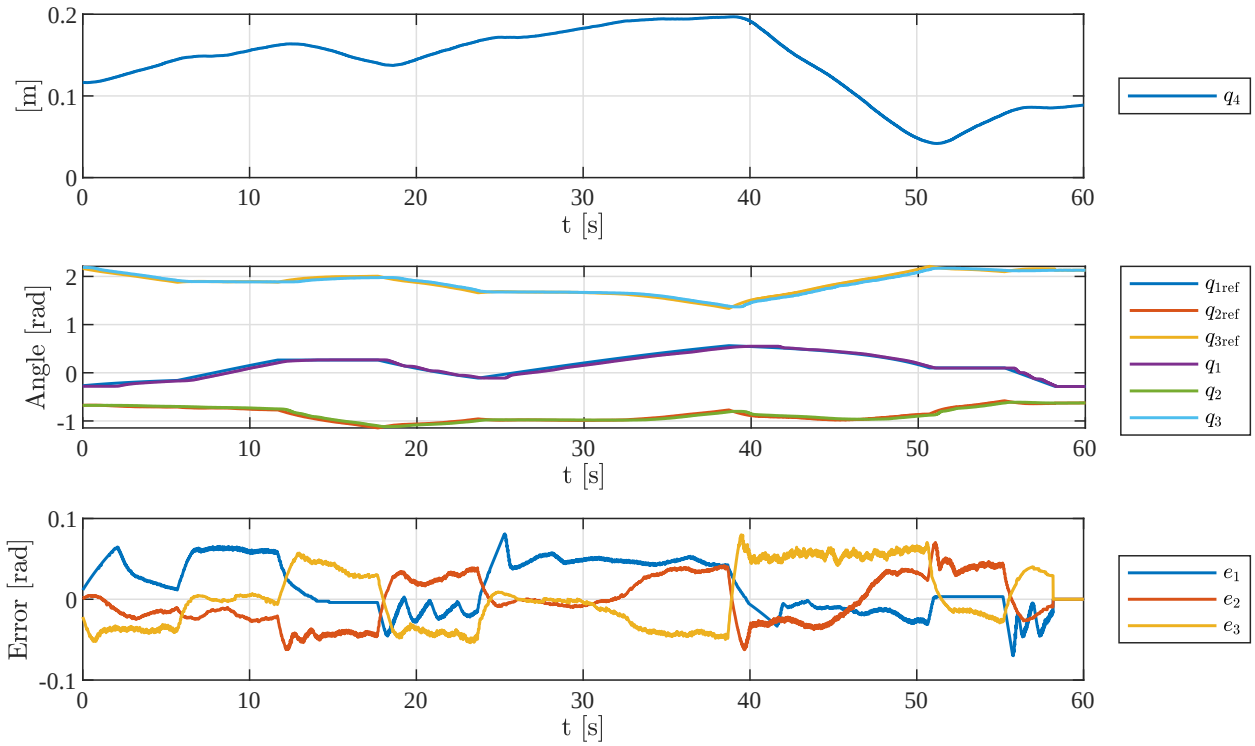

Figure 7.23: Joint Results PID - Controller 2

7.3.8 PID Controller 3

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
4	7	5	1	1	2	0.1	0.1	0.2	0.05	10·0.05

Table 7.8: Gains PID - Controller 3 With Telescope

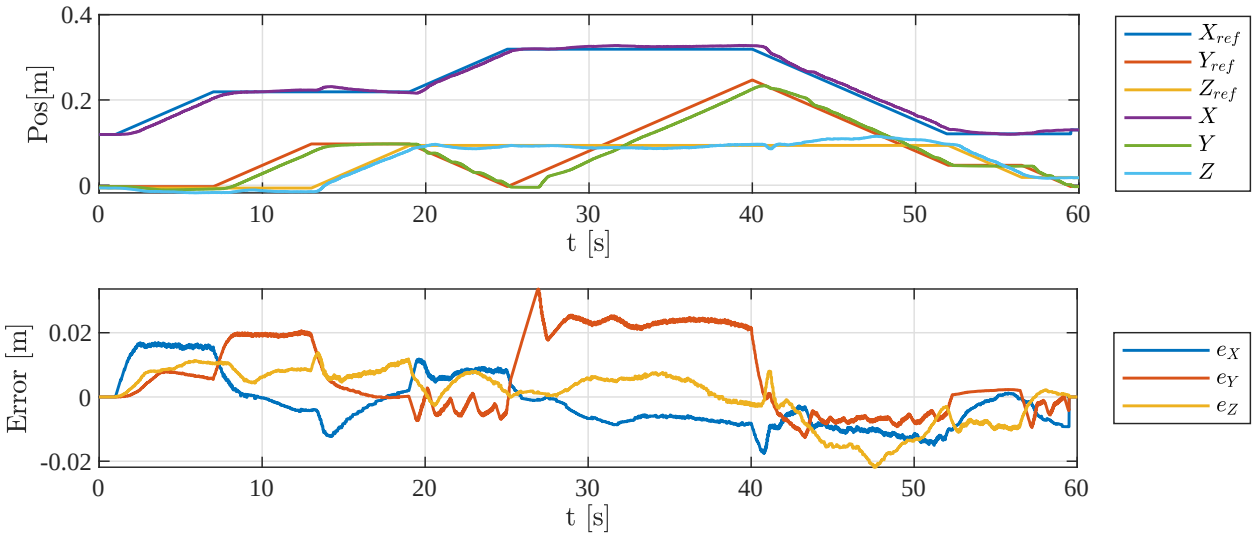

Figure 7.24: Cartesian Results PID - Controller 3

For the PID - Controller 3, all the P gains are reduced, and integral K_{I3} is increased.

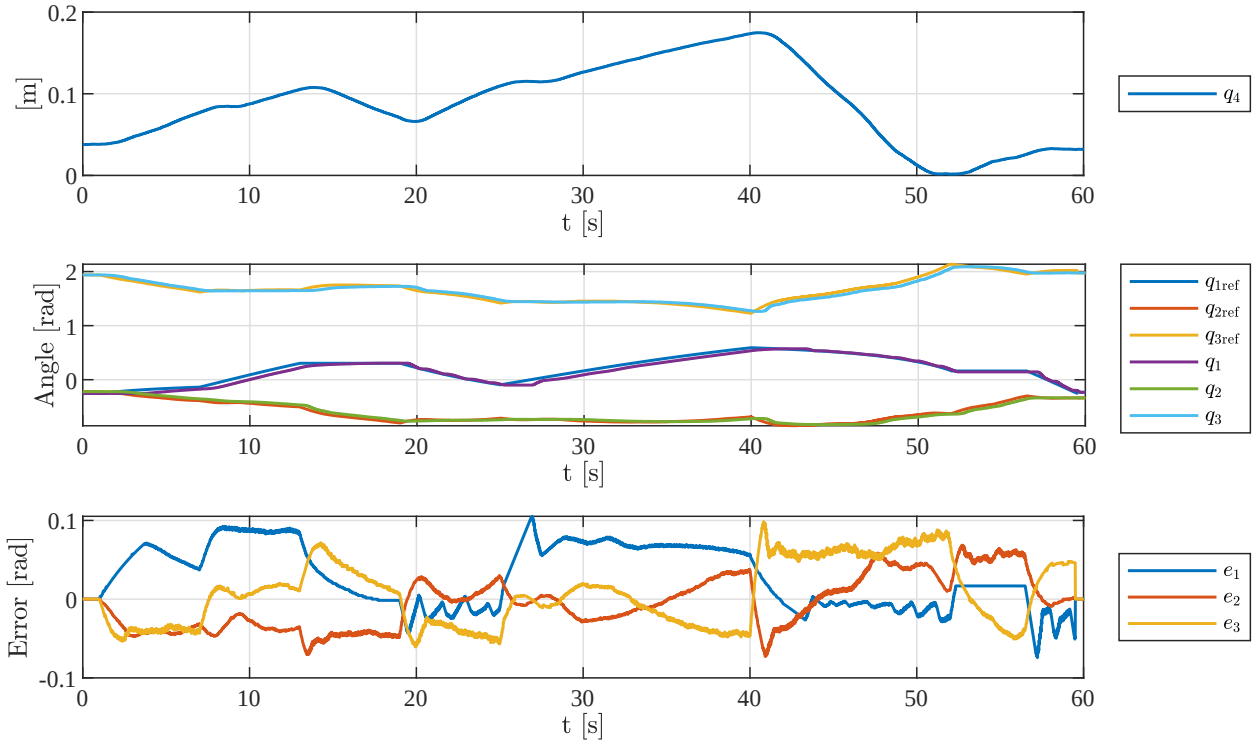

Figure 7.25: Joint Results PID - Controller 3

7.3.9 PID Controller 4

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
2	3	3	2	2	4	0.1	0.1	0.2	0.05	10·0.05

Table 7.9: Gains PID - Controller 4 With Telescope

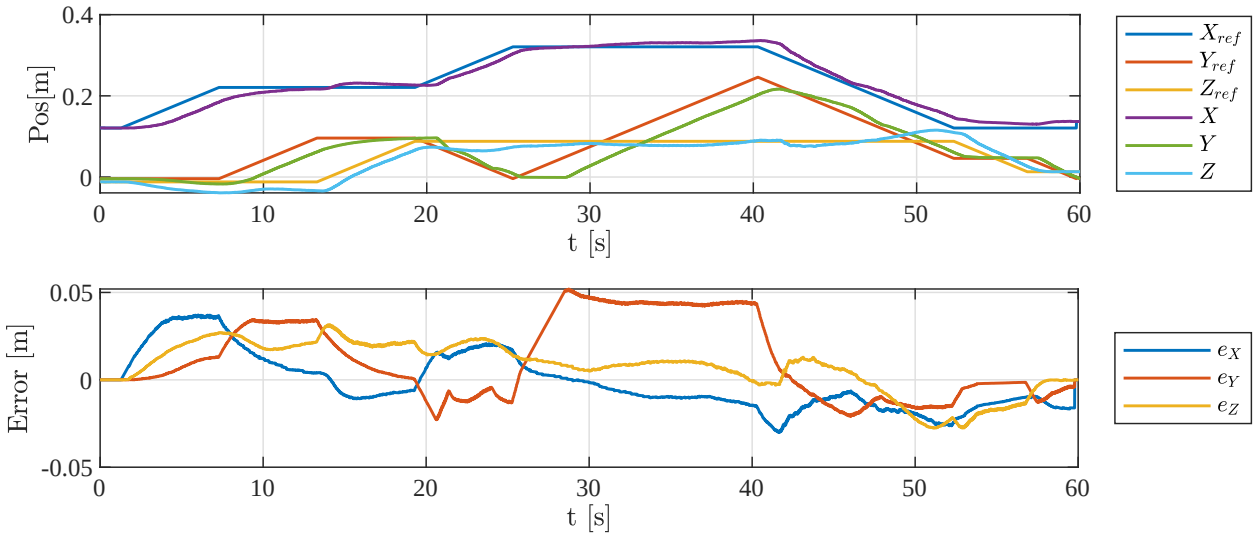

Figure 7.26: Cartesian Results PID - Controller 4

Reduction in all the P gains, and increasing all the I gains shows very few oscillation. However the error is large as expected.

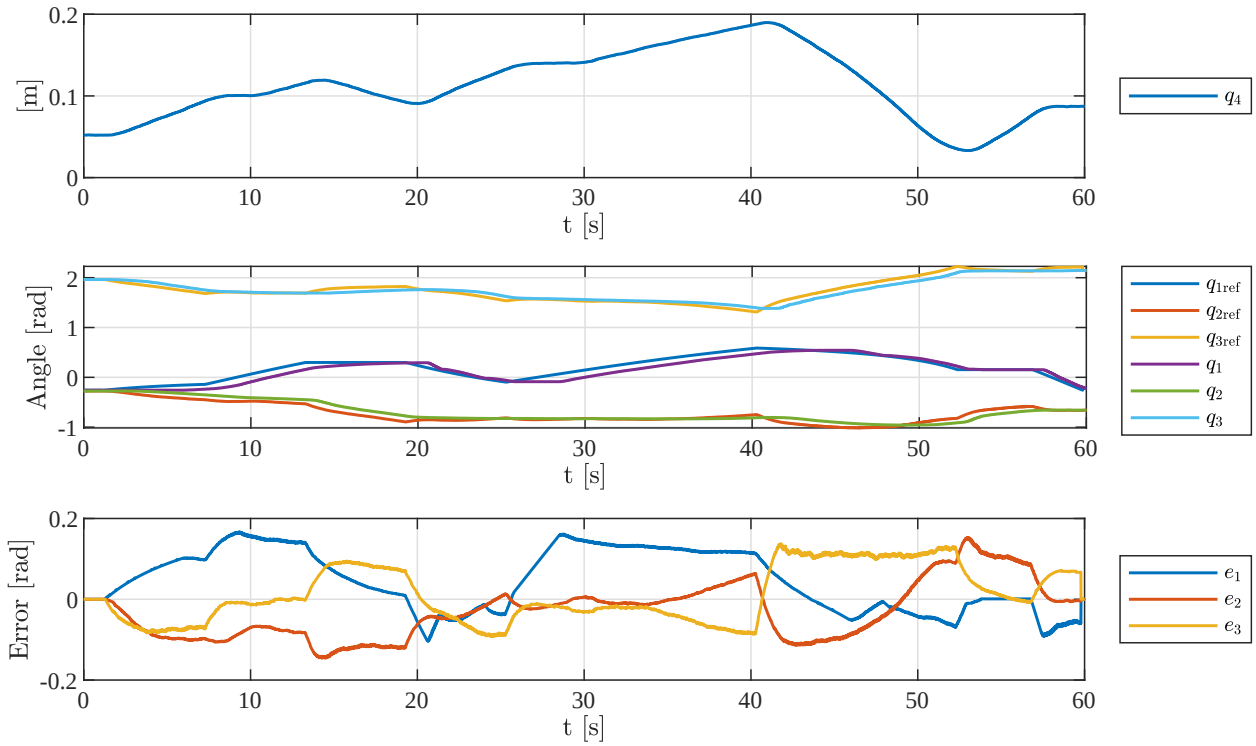

Figure 7.27: Joint Results PID - Controller 4

7.3.10 PID Controller 5

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
4	5	6	2	2	4	0.05	0.05	0.05	0.05	10·0.05

Table 7.10: Gains PID - Controller 5 With Telescope

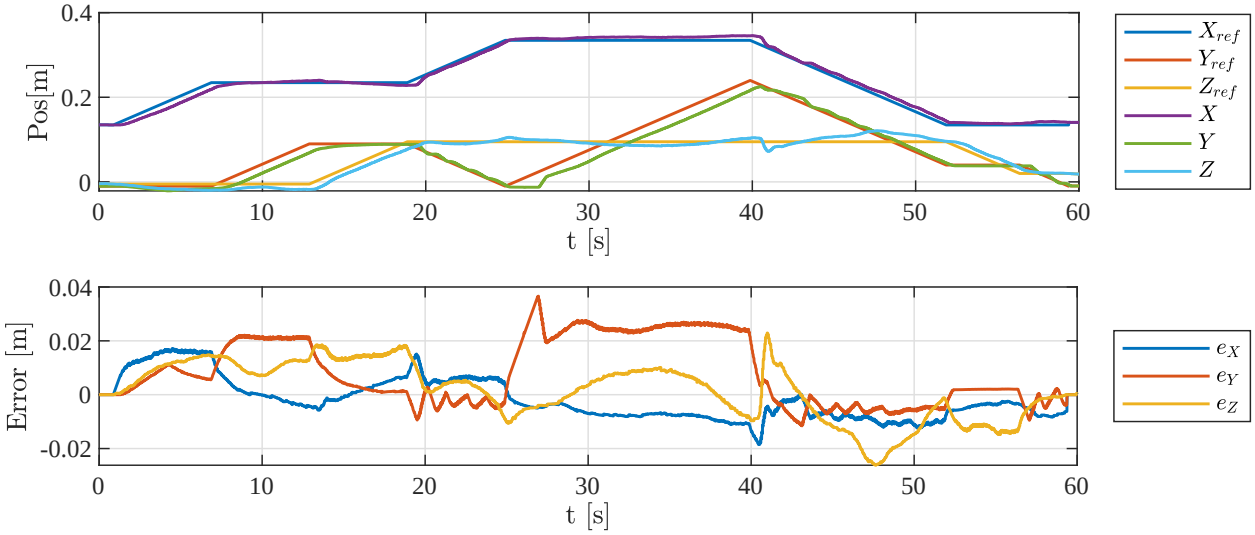

Figure 7.28: Cartesian Results PID - Controller 5

Increasing the P gains and reducing the D gains reduces the error.

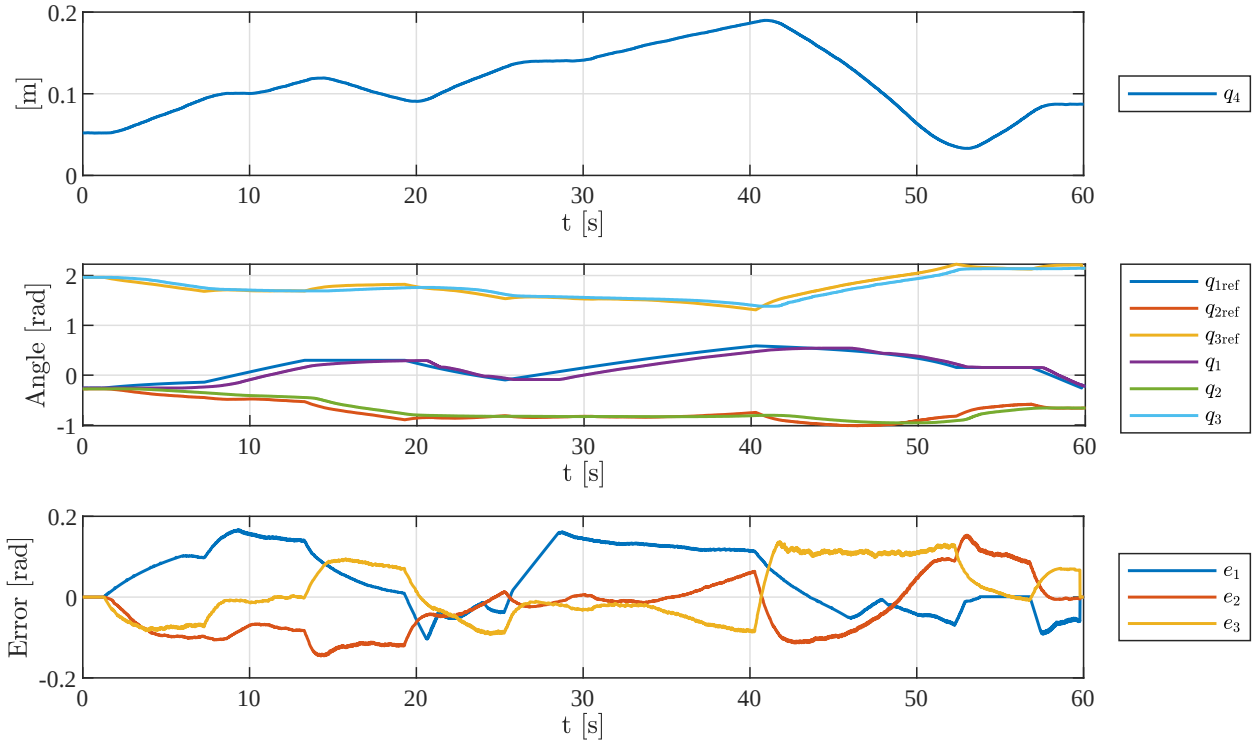

Figure 7.29: Joint Results PID - Controller 5

7.3.11 PID Controller 6

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	6	1	1	1	0.05	0.05	0.05	0.05	10·0.05

Table 7.11: Gains PID - Controller 6 With Telescope

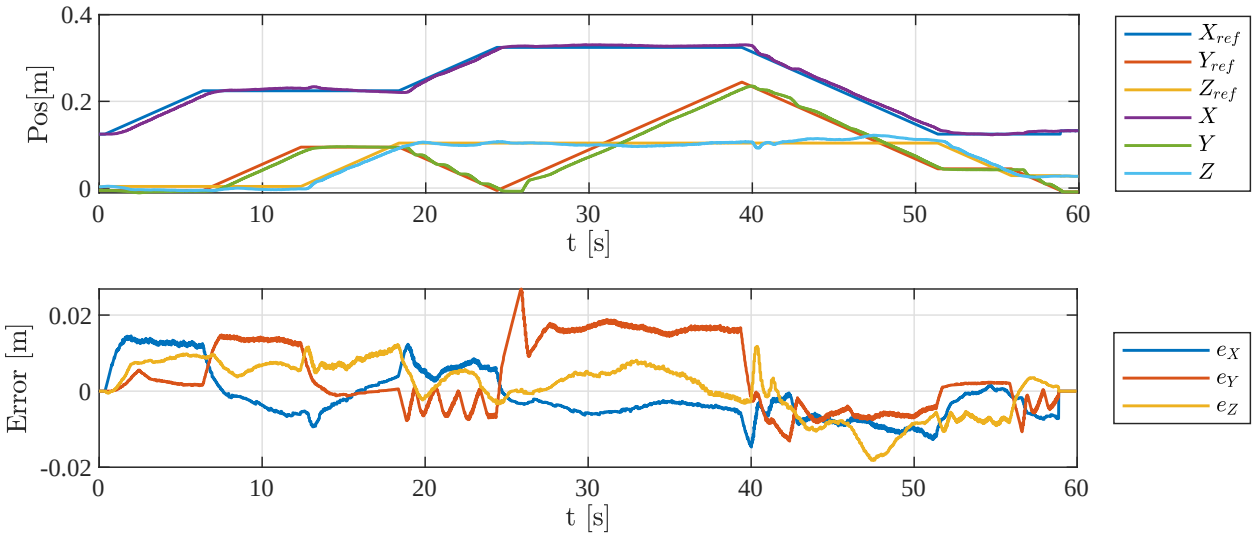

Figure 7.30: Cartesian Results PID - Controller 6

Going back to the P gains and I gains from PID controller 2, shows similar results to PID Controller 2, but less frequency and larger amplitude on oscillation as expected.

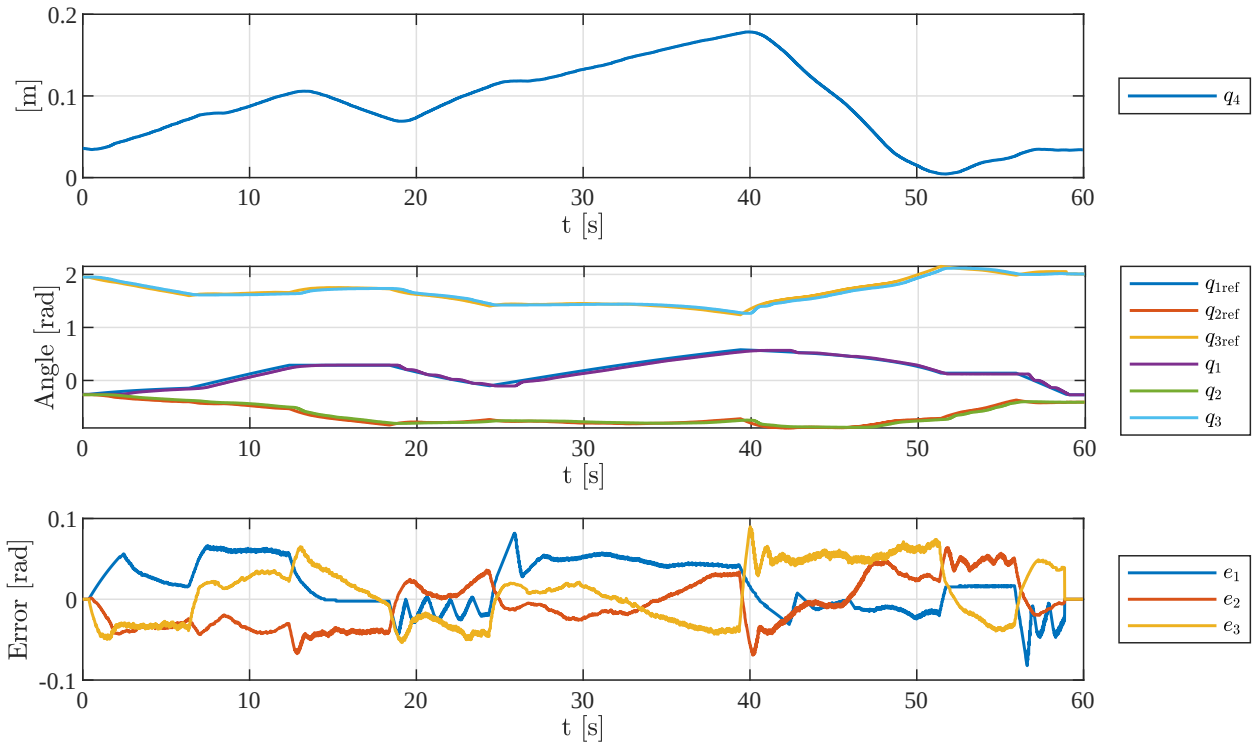

Figure 7.31: Joint Results PID - Controller 6

7.3.12 PI Controller 1

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	6	2	2	2	0	0	0	0.05	10·0.05

Table 7.12: Gains PI - Controller 1 With Telescope

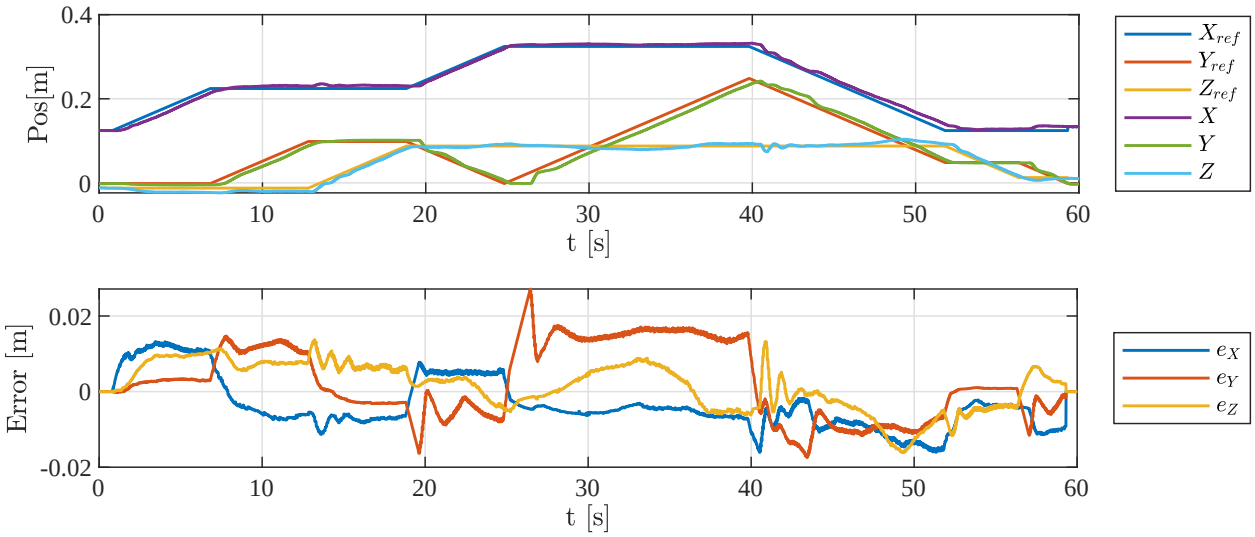

Figure 7.32: Cartesian Results PI - Controller 1

Removing the D part and increase the integral gives larger error but less oscillations.

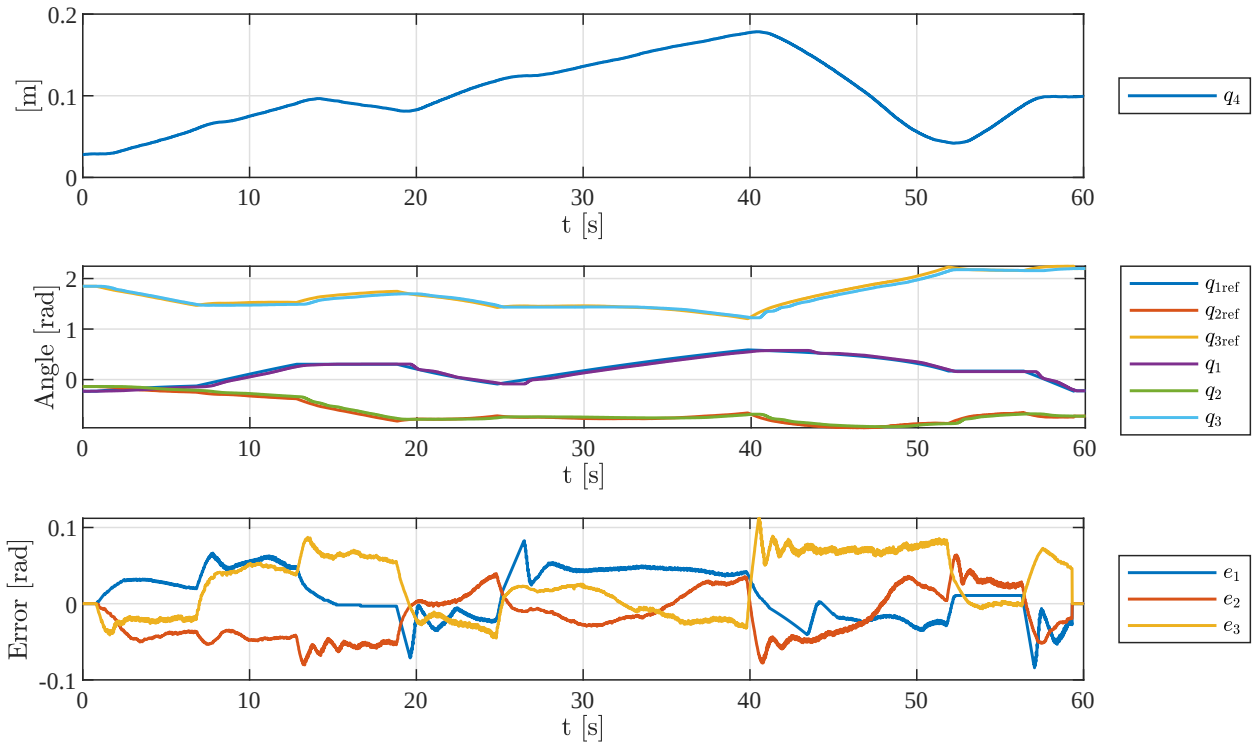

Figure 7.33: Joint Results PI - Controller 1

7.3.13 PI Controller 2

K_{P1}	K_{P2}	K_{P3}	K_{I1}	K_{I2}	K_{I3}	K_{D1}	K_{D2}	K_{D3}	$K_{\text{telescopeOUT}}$	$K_{\text{telescopeIN}}$
5	8	6	3	3	3	0	0	0	0.05	10·0.05

Table 7.13: Gains PI - Controller 2 With Telescope

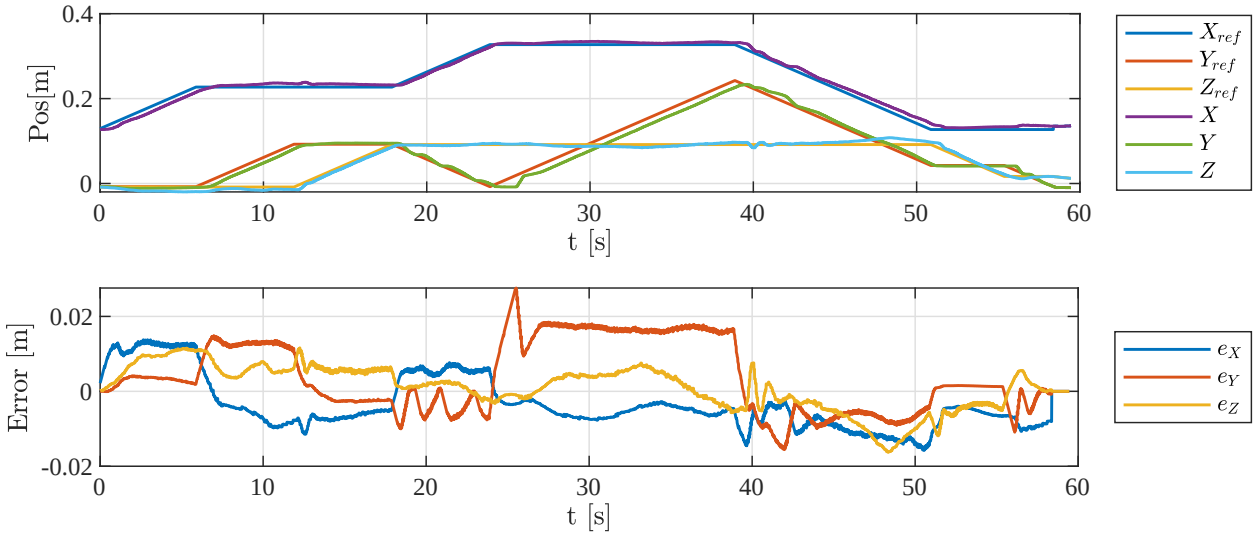

Figure 7.34: Cartesian Results PI - Controller 2

Increasing the integral term recuses the error in some parts.

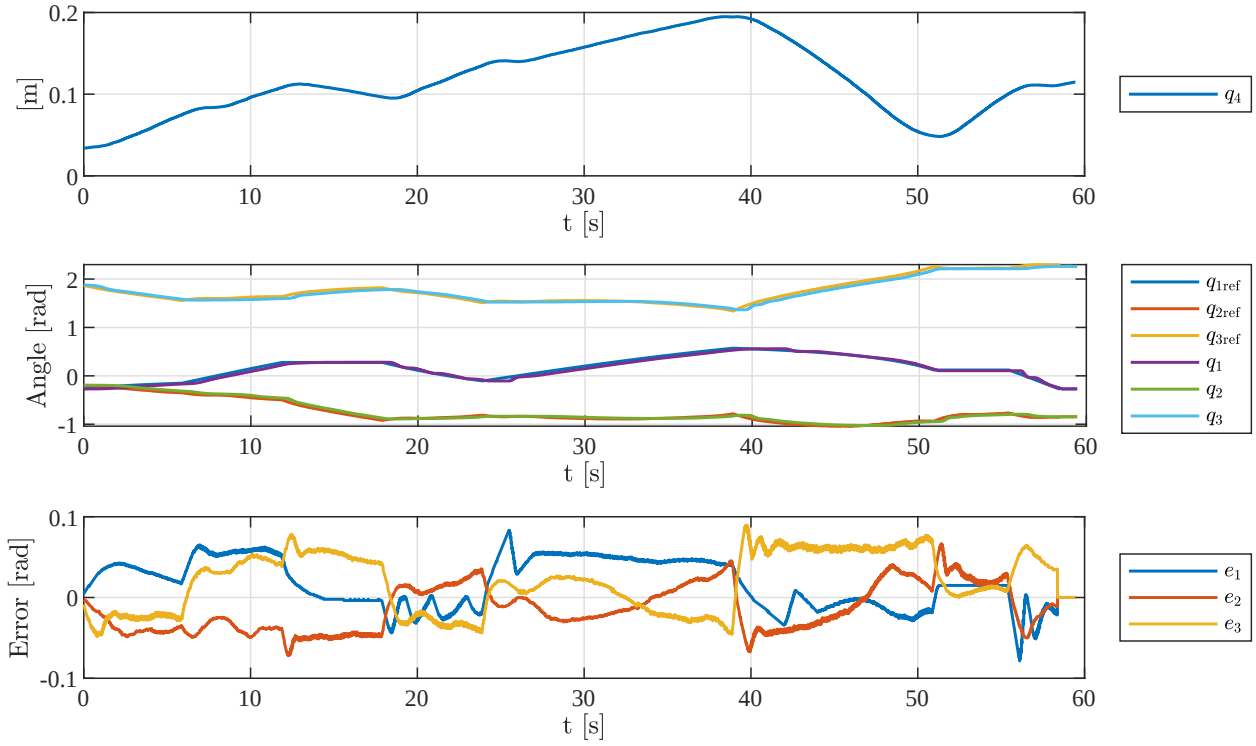

Figure 7.35: Joint Results PI - Controller 2

7.3.14 PID Tuning Wrap Up

For the controller tuning above, different controllers had different advantages and disadvantages. The system is non linear, and different controllers fit the system with for different crane configurations. For example, if the telescope is at full extension, the joints acts differently than if it is fully retracted. Because the dynamics of the system is neglected, the controller best suited over all is chosen as the acting controller, and is used Cartesian Control both with and without heave compensation. More testing and tuning could have been done as well, but the all over best acting controller from these experiments was PID Controller 6.

7.4 Heave Compensation

Using PID - Controller 6, the heave compensation is tested. The handle is provided with random manually actuated "waves".

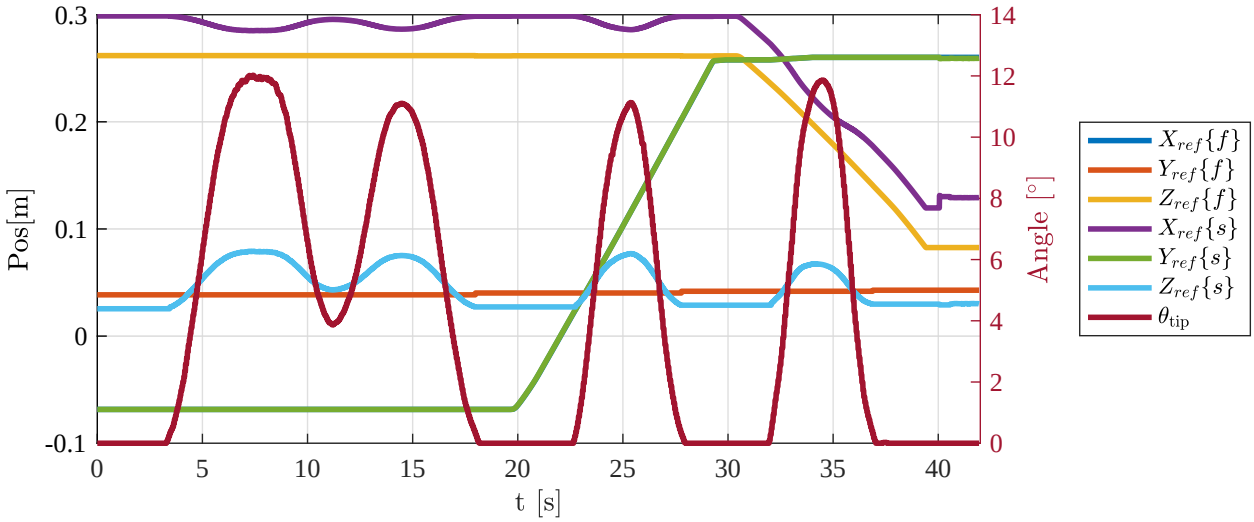

Figure 7.36: Cartesian Reference Relationship

Figure 7.36 shows the Cartesian position reference in frame coordinate {f}, the crane coordinate {s}, and the angle between them θ_{tip}. The reference for the control system is the crane coordinate {s}, and the reference signal compensate directly for the heave as illustrated in the figure.

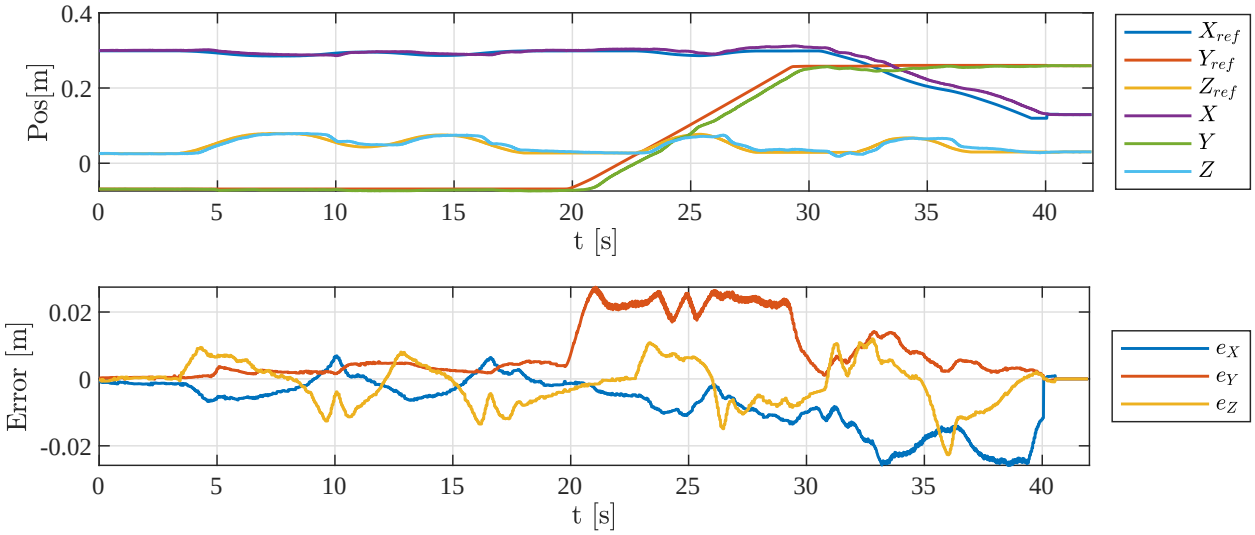

Figure 7.37: Cartesian Results Heave

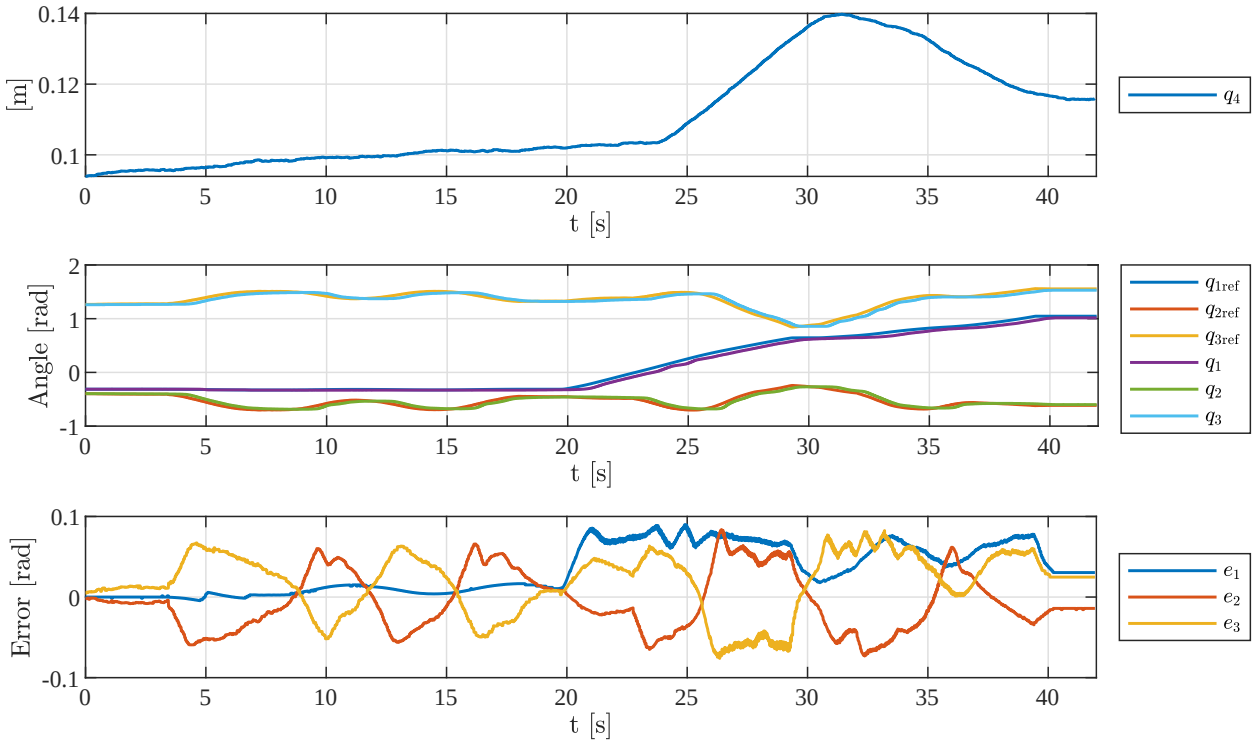

Figure 7.38: Joint Results Heave

The results show that the crane is following the compensated reference signal. However, the error in X, and Z direction is affected by the tilt θ_{tip} as expected.

Chapter 8

Discussions

This chapter will discuss the results from the hardware and different controllers before pointing out some challenges and introduce some potential further work.

8.1 Hardware Result

The look of the demo model fits the demonstration and education purpose by having visible components, as well as a clean design. The model is a plug and play solution, meaning the only connection needed is the 230 [V] supply, and the model can be used.

The manual tilt handle is meant to give the user a more understandable experience of heave compensation. By manually give a disturbance of your choice, and see if the crane can handle it is interesting. Especially if the movement is to quick for the crane to compensate fast enough, this gives the user an understanding of the limitations in a mechatronics system.

8.2 Controller Result

8.2.1 Manual Controller

The results from the manual controller reflects the relationship between the servo motor position, to the valve opening, and the joint velocity. Because the sensors are mounted directly on the joint, the joint velocity is compared directly against the valve opening. The results from q_1 show similar amplitude with long telescope and short telescope. However, the start of motion is different. This is a challenge regarding the mapping of the manual controller. The joy stick is mapped to jump directly to a certain value, corresponding to the start of motion, but when the start of motion varies, the joy stick mapping will not be perfect for all configurations.

For q_2 however, the start of motion is similar, but there is a larger difference in the magnitude of the angular velocity between the joints. For this joint there is also a more significant difference in magnitude of the angular velocity, depending on if the cylinder is hoisting or lowering. If the same flow is flowing constant, the cylinder would retract faster than extracting, because the area on the rod side is smaller. Also, because the gravity is working the same direction as the actuator while lowering, this could also cause higher lowering angular velocity.

q_3 also shows similar results to q_2. Here it is more clear that the hoisting starts with a larger valve opening for the configuration of long telescope.

For q_4, based on the hydraulic assumptions, that the flow is the same in both directions the cylinder should retract faster than extracting. However, the results shows the opposite. The telescope is extracting faster than retracting, for both configurations, also the one pointing up (with gravity working opposite). For this case, it is the valve and servo assembly not centered, meaning the servo value is not corresponding exactly to the valve opening, thus more servo actuation in one direction is needed to have the same flow in both.

8.2.2 Cartesian

For the Cartesian controller, the same Cartesian path is followed for all test. Other path could have revealed other weaknesses with the control system, however for comparative reason, the same path was used consequently.

First the locked telescope 3 DOF P - Controller shows a maximum position error of approximately 2 [cm]. This is not so bad, considering the non linear characteristics of the actuators presented in the manual controller. Activating the telescope with the same controller shows close to similar results for the first part of the path, but much more reach. Naturally, with a long telescope extension, the error tends to be larger, and having larger oscillations.

The combined kinematics solution is removing the actuator redundancy. The 3 DOF inverse position kinematics, with variable q_4 worked as indented, showing good results. To be able to adjust the telescope participation, also depending on the direction of $\dot{q_4}$ worked for the presented experiments. Because the combined kinematics worked as intended, does not mean it is the best algorithm for controlling q_4 in parallel. Other algorithms could be used to control q_4, depending on the application. For example, the motion of q_4 could be dependent on the distance from the origin of the crane or the position within the workspace.

Adding the derivative gain reduces the amplitude of the oscillations, but increases the frequency. Therefore, the balance is found with a low derivative gain for all joints. Adding integral part helps to remove steady state error. The resultant PID controller presented as PID - Controller 6, only have one spike with end effector error over 2 [cm] in Cartesian space. This error occurs when q1 is actuated, with the telescope half extended. This turned out to be a challenge for all the controllers, and is because of two things. Firstly, q_1 have a small spectrum of actuation from no motion to full motion, as presented in manual results. Also, having some extension of telescope, and velocity reference close to perpendicular to the direction of the crane, makes a high moment of inertia.

Actuator and workspace limits are also tested in the Cartesian mode. By trying to move the end effector outside the desired workspace, the velocity reference = 0, until the direction is changed to the opposite. If the actuators reach close to the end position they also stops until the direction of motion changes to the opposite direction. The reason this is only implemented in Cartesian mode, is that if some sensors are giving false values or not working, it should always be possible to use the crane in manual mode.

Both for manual controller and Cartesian controller shows significant difference depending on the crane configuration. Probably the difference is significantly larger with load on the crane as well. Therefore, a pure position controller can work, but is not ideal. Ideally some controller taking reacting forces, flow rate etc into account, would make it easier to make a more accurate controller.

8.2.3 Heave Compensated

The heave compensation control shows similar results to the Cartesian controller. Now the reference signal changed based on the external disturbance. The actual position is lagging some to the reference position, thus the error changes direction based on the direction of the disturbance. The

heave compensation is absolutely proving the concept. However, what is not presented in the plots is the delay from the real world, to the angle θ_{tip} is reaching the IPC. This is because the moving average filter inside the Arduino software is introducing the delay.

8.3 Challenges

8.3.1 Hardware Implementation

Hardware implementation is time consuming, however high reward for the learning aspect of the student. Designing and actually building the hardware makes a good platform for testing concepts as well as demonstrating different mechatronics aspects. One of the most time consuming challenge was the I2C communication between the sensor and the input Arduino. The sensors are mounted with different length from the Arduino. The sensors showing the most problem was the MPU9250. The challenge was noise, probably causing one or more bits in the data do be flipped, and crash the I2C bus.

Many different measures were carried out, such as have different Arduino for inputs and outputs, have different power sources for motors and controller, shielded cables, filters and ferrite cores, separate power cables from communication cables etc. Most of the measures probably helped, however the shielded cable, together with the common ground between the high and low voltage connection was the major difference. From then, only a couple of I2C crashes occurred, meaning the sensors, Cartesian controller and Heave compensation works satisfactory.

Another solution to this problem could be to change the MPU to a rotary encoder mounted on the revolute joint of the tilt. However, the reason this was not done for this project is that MPU would be more natural to use for a real application such as an offshore vessel, and because this is a demonstration model, it is an advantage to be able to show the MPU and explain how it works and why it is used.

8.3.2 Neglecting Dynamics

Because the crane and hydraulic system is small, the option for sensors regarding flow and pressure is limited. Therefore, for the scope of this project the dynamics of the system was neglected from the beginning. However simple it sounds, I think having the sensors and consider the dynamics could give advantages as well. The actuator redundancy could have one more factor to use in the inverse kinematics solving, and the flow could make more accurate velocity control of the joints.

8.4 Further work

For the further work of this demo crane, my suggestion is to look into the possibility to purchase or develop small sensors for the hydraulic system. This combined with a better controller using flow or/and pressure, could make a lot more accurate tool point control.

Better filtering/sensor fusion for the MPU9250, to avoid the large time delay affecting the heave compensation.

Also, it could be interesting to have a monitor connected to the demo model directly, to eliminate the need of a laptop to have the HMI. The monitor could also be used for visualization of the crane model. This could be interesting to demonstrate the similarities between the digital environment and the hardware.

Chapter 9

Conclusion

To sum up the project, the overall goal of developing and building a demo model is successfully archived. The demo model have a CAD model, including the parts used for the actual model. Appropriate sized sensors are attached and used to successfully collect data from four DOF, q_1 - q_4.

The forward and inverse kinematics is derived for the knuckle boom crane and simulated in Matlab Simulink. Due to the actuator redundancy, a combined solution removing the actuator redundancy is found, and tested. The solution works in simulation as well as on the hardware tests.

Sensors are interfaced successfully with the Beckhoff IPC. Different control modes are implemented, such as Manual mode, Cartesian controller, and Heave compensated controller. The results from the Cartesian controller shows maximum end effector position error of around 2 [cm], for a relatively slow control system without load. This controller could be tuned more, or even hydraulic sensors for flow and pressure could be included to introduce more aspects for a better controller.

The project execution went according to the desired plan. All the objectives where reached. From the beginning, it was already a tight schedule, because of all the necessary hardware. However, by putting in some extra long days in the building, implementation and testing phase, the development and implementation was ended by mid April, to be able to focus on the report and the results from then.

Bibliography

[1] *12-Bit Programmable Contactless Potentiometer*. AS5600. Rev. 1-06. AMS. June 2018.

[2] AB-T-Srl. *EasyCAT Shield per Arduino*. URL: https://www.bausano.net/it/hardware/arduino-easyca htmls. (accessed: 10.01.2024).

[3] *Adafruit PCA9685 16-Channel Servo Driver*. PCA9685. Rev. Date: 2024-03-14 04:09:17. Adafruit. Mar. 2024.

[4] Ashay Aswale Apurva Patil Maithilee Kulkarni. "Analysis of the inverse kinematics for 5 DOF robot arm using D-H parameters." In: *ieeexplore* (12 March 2018). DOI: https://ieeexplore.ieee.org/ab

[5] Arduino. *Libraries*. URL: https://www.arduino.cc/reference/en/libraries/. (accessed: 24.05.2024).

[6] *Arduino® MEGA 2560 Rev3*. MEGA2560. Rev. 3. Arduino. Apr. 2024.

[7] *Arduino® UNO R3*. UNOR3. Rev. 2. Arduino. Nov. 2023.

[8] automaticaddison. *Denavit–Hartenberg parameters*. URL: https://automaticaddison.com/homogeneous-transformation-matrices-using-denavit-hartenberg/. (accessed: 13.02.2024).

[9] BECKHOFF. *CX5240 Embedded PC with Intel Atom processor*. URL: https://www.beckhoff.com/en-en/products/ipc/embedded-pcs/cx5200-intel-atom-x/cx5240.html?. (accessed: 1.05.2024).

[10] BECKHOFF. *EK1818 EtherCAT Coupler with integrated digital inputs/outputs*. URL: https://www.beckhoff.com/en-en/products/i-o/ethercat-terminals/ek1xxx-bk1xx0-ethercat-coupler/ek1818.html?. (accessed: 1.05.2024).

[11] BECKHOFF. *EL3255 EtherCAT Terminal, 5-channel analog input, potentiometer, ...* URL: https://www.beckhoff.com/en-en/products/i-o/ethercat-terminals/el3xxx-analog-input/el3255.html. (accessed: 1.05.2024).

[12] Beckhoff. *TwinCAT automation software*. URL: https://www.beckhoff.com/en-en/products/automation/twincat/?pk_campaign=AdWords-AdWordsSearch-TwinCAT_EN&pk_kwd=twincat&gad_source=1&gclid=Cj0KCQjwltKxBhDMARIsAG8KnqXQBIl2rADk0Hs3mgivMGNwNQE_cJ-Erw7NThtz-wipgwcB. (accessed: 3.05.2024).

[13] L. Beiner. "Minimum-Force Redundancy Control of Hydraulic Cranes." In: *IFAC Proceedings Volumes* (1997). DOI: https://www.sciencedirect.com/science/article/pii/S1474667017443254#bi

[14] John Deere. *Find the Forwarder That Works For You*. URL: https://www.deere.com/en/forwarders/. (accessed: 05.05.2024).

[15] J. Denavit and R. S. Hartenberg. In: *Trans. ASME E, Journal of Applied Mechanics* ().

[16] Daniel Hagen. *hagenmek*. URL: https://github.com/hagenmek. (accessed: 23.05.2024).

[17] Morten K. Ebbesen Konrad J. Jensen and Michael R. Hansen. "Development of 3D Anti-Swing Control for Hydraulic Knuckle Boom Crane." In: *MIC journal* (2021). DOI: https://www.mic-journal.no/

[18] Kevin M Lynch and Frank C. Park. *Modern Robotics: Mechanics, Planning, and Control*. English (US). Cambridge Univeristy Press, 2017. ISBN: 978-1107156302.

[19] MathWorks. *pinv Moore-Penrose pseudoinverse*. URL: https://se.mathworks.com/help/matlab/ref/pinv.html. (accessed: 18.04.2024).

[20] leimbach modellbau.de. *Klappladekran*. URL: http://leimbach-modellbau.de/Produkte/LKW-Aufbauten/Klappladekran/09401(T)/. (accessed: 10.01.2024).

[21] leimbach modellbau.de. *LangholzLadekrann*. URL: http://leimbach-modellbau.de/Produkte/LKW-Aufbauten/Holzladekrane/09410(T)/. (accessed: 26.04.2024).

[22] *MPU-9250 Product Specification Revision 1.1*. VL53L0X. Rev. 1.1. InvenSense. June 2016.

[23] NxRLab. *Modern Robotics: Mechanics, Planning, and Control Code Library*. https://github.com/NxRLab/ModernRobotics/tree/master. 2017.

[24] OpenAI. *ChatGPT*. Version GPT-4. 2024. URL: https://chatgpt.com/.

[25] Richard P. Paul. *Robot Manipulators: Mathematics, Programming, and Control*. 1st. Cambridge, MA, USA: MIT Press, 1982. ISBN: 026216082X.

[26] PerHenrikHardeberg. *UIADemoCrane*. URL: https://github.com/PerHenrikHardeberg/UIADemoCrane. (accessed: 23.05.2024).

[27] Valerio Salvucci et al. "Comparing Approaches for Actuator Redundancy Resolution in Biarticularly-Actuated Robot Arms." In: *IEEE/ASME Transactions on Mechatronics* 19.2 (2014), pp. 765–776. DOI: 10.1109/TMECH.2013.2257826.

[28] *TCA9548A Low-Voltage 8-Channel I 2C Switch with Reset*. TCA9548A. Rev. G. Texas Instruments. May 2012.

[29] *Time-of-Flight ranging sensor*. VL53L0X. Rev. 5. Life Augmented. Dec. 2022.

[30] Trello. *Trello brings all your tasks, teammates, and tools together*. URL: https://trello.com/. (accessed: 01.09.2016).

[31] wikipedia. *Denavit–Hartenberg parameters*. URL: https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters. (accessed: 13.02.2024).

[32] Wikipedia. *Excavator controls*. URL: https://en.wikipedia.org/wiki/Excavator_controls. (accessed: 05.05.2024).

[33] Jun Wu et al. "Dynamics and control of a planar 3-DOF parallel manipulator with actuation redundancy." In: *Mechanism and Machine Theory* 44.4 (2009), pp. 835–849. ISSN: 0094-114X. DOI: https://doi.org/10.1016/j.mechmachtheory.2008.04.002. URL: https://www.sciencedirect.com/science/article/pii/S0094114X08000839.

[34] Huafeng Ding Zhen Huang Qinchuan Li. *Theory of Parallel Mechanisms, 2013, Volume 6, Chapter: Basics of Screw Theory*. Volume 6. 2013. ISBN: 9789400742000.

Appendix A

Start Up Guide For Demo Model

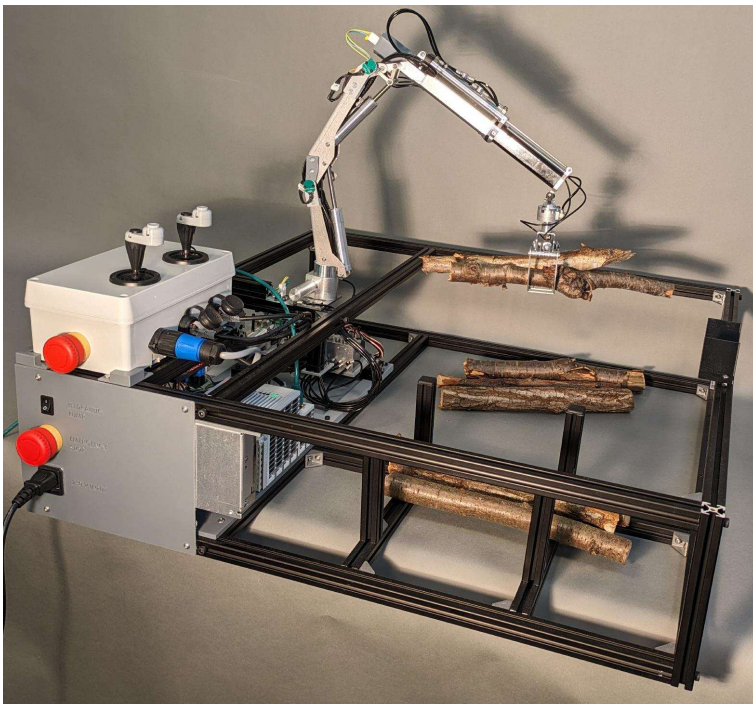

Figure A.1: Demo Model

A.1 Step by Step Without HMI

1. Connect the 230 [V] power supply, make sure the hydraulic pump is turned of and release the emergency stop. See section A.3.
2. Switch to Manual mode, move the crane to rest position. See section A.3.
3. Reset Input Arduino, to reset sensors. See section A.4.
4. Ready to use Manual And Cartesian Mode. See section A.5.
5. If sensor communication is lost the Cartesian Mode will not work. Then do the reset step 6 - 8 again.

A.2 Step by Step With HMI

1. Make sure to have Twincat 3, with project DemoCraneControl.sln on your laptop. If not, see section A.6, OR Connect to the to the Beckhoff PLC with your laptop and open project from target.
2. Connect the 230 [V] power supply, make sure the hydraulic pump is turned of and release the emergency stop. See section A.3.
3. On your laptop (see section A.6): Make sure target is CX-6DB777.
4. Switch to Manual mode, move the crane to rest position. See section A.3.
5. Reset Input Arduino, to reset sensors. See section A.4.
6. Ready to use Manual And Cartesian Mode. See section A.5.
7. If sensor communication is lost, do the reset step 6 - 8 again.
8. For using Heave Compensation or Path Generation, open HMI and use as explained in section A.6.

A.3 Connections

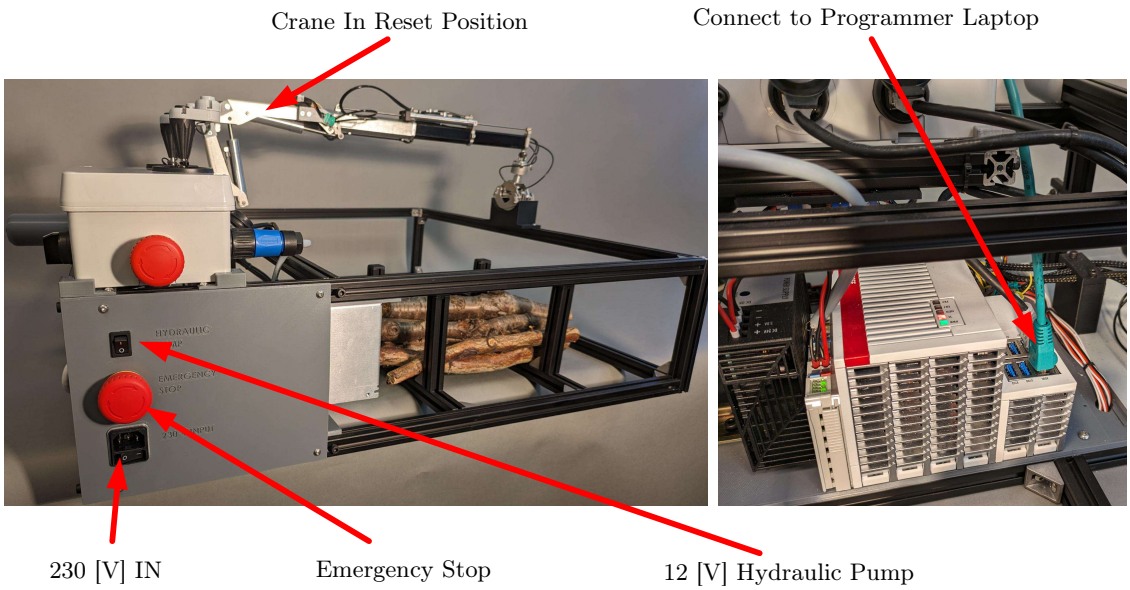

Figure A.2: Buttons and Connections

A.4 Reset Sensors

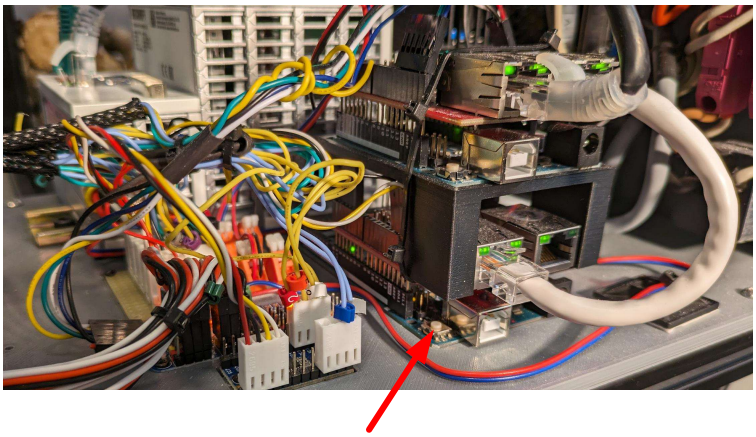

Figure A.3: Reset Input Arduino

Make sure the crane is in reset position before pressing reset button.

A.5 Different Modes Control

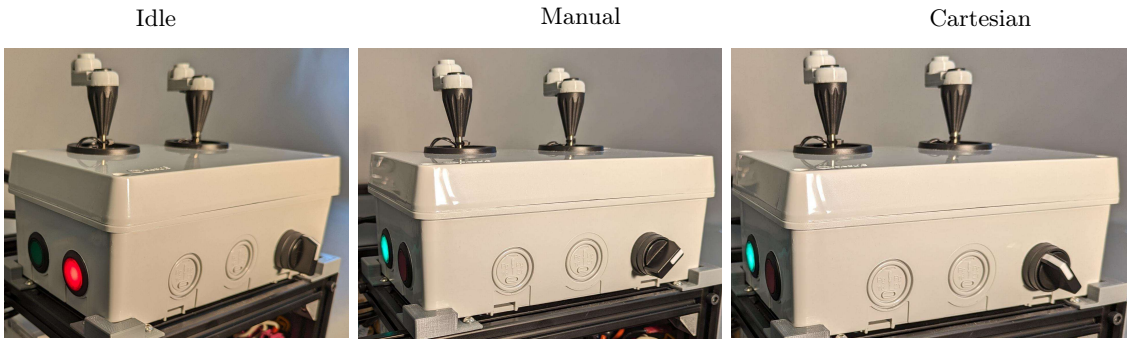

Figure A.4: Different Modes

Manual Mode Control

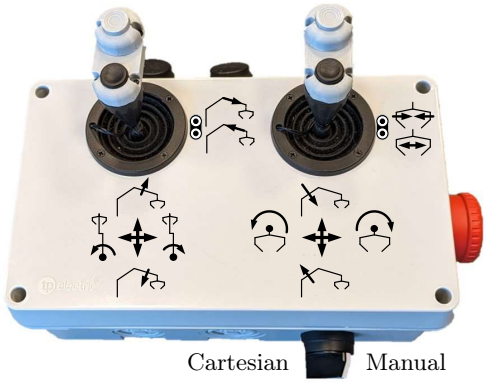

Figure A.5: Manual Crane Control

Cartesian Mode Control

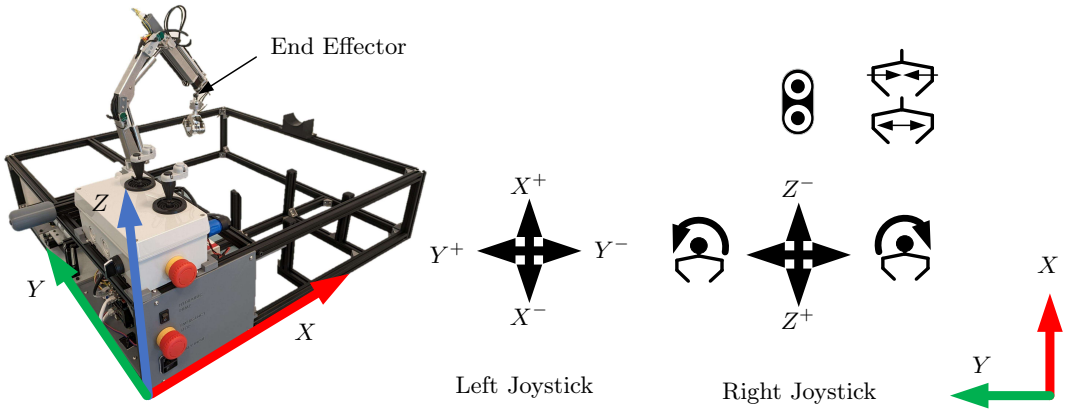

Figure A.6: Cartesian Crane Control

When Cartesian Mode is active, Heave Compensation is available. Enable Heave Compensation in HMI.

A.6 PLC Program

Software can be downloaded from public github repo:
https://github.com/PerHenrikHardeberg/UIADemoCrane.git

Figure A.7: Add Slaves TwinCAT

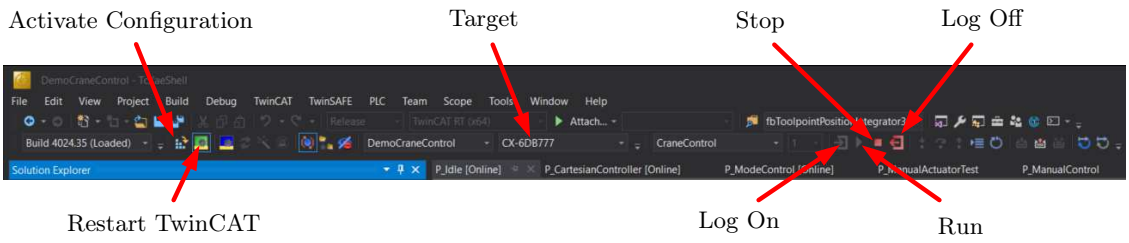

Figure A.8: PLC Log On

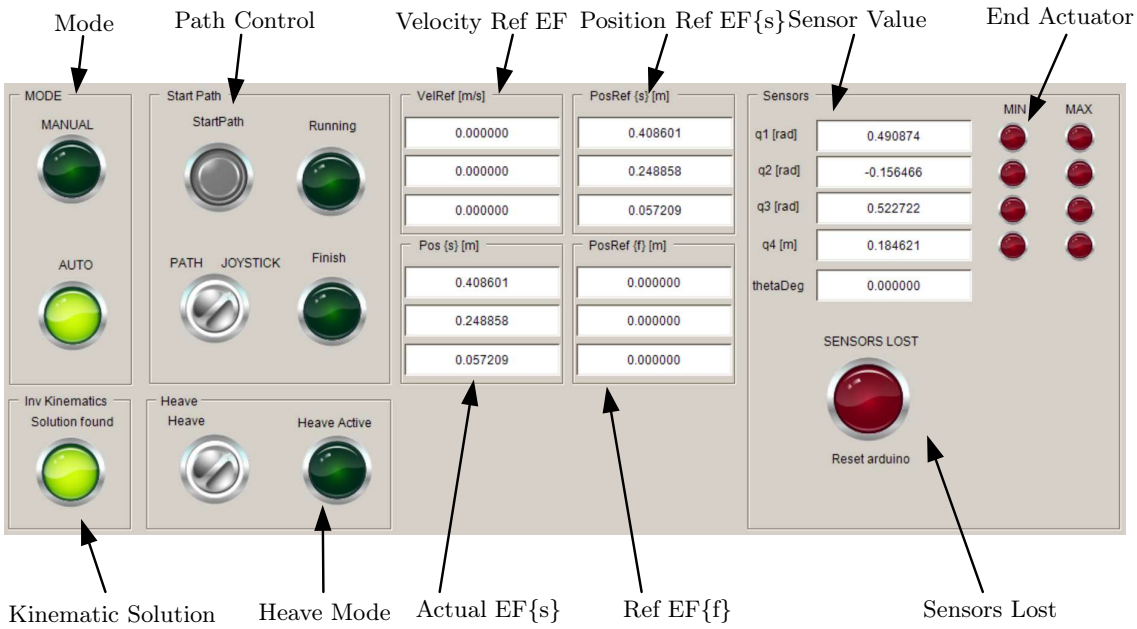

Figure A.9: HMI

If sensors are lost, reset input Arduino.

Appendix B

Appendix Code

B.1 INPUT_SLAVES

```
#define CUSTOM // Define custom variable is used
#include "INPUT_SLAVE.h"


// 0x70 is i2c multiplexer
// Encoders:
#include <AS5600.h>

// Define three as5600 objects
AS5600 stpA;
AS5600 stpB;
AS5600 stpC;


#include "EasyCAT.h"                // EasyCAT library
#include <SPI.h>                    // SPI library - The EasyCAT ...
    shield is connected to the SPI bus
EasyCAT EASYCAT;                    // EasyCAT istantiation
#include <Wire.h>                   // I2C library

//Global Copied from IO Script EasyCat
unsigned long Millis, PreviousMillis = 0;
unsigned char EcatState;
unsigned char N_Cycles = 0;
bool Blink = true;
UWORD Raw;


// ADDED 12.03.2024 TOF Sensor VL53lXX
#include "Adafruit_VL53L0X.h"
Adafruit_VL53L0X lox = Adafruit_VL53L0X();

//////////////////////////////////////////////////////////////////
/// MOVING AVARAGE FILTER_TEST:
#include "MovingAverageFloat.h"
MovingAverageFloat <32> Floatfilter; // For Laser sensor

MovingAverageFloat <32> Floatfilter2; // For IMU ANGLE Value
//////////////////////////////////////////////////////////////////
```

```
//INITIATES  SENSORS
float radA = 0.0;
float radB = 0.0;
float radC = 0.0;
float xRot = 0.0;
float yRot = 0.0;
float TelescopeDistance_mm = 0.0;

// Encoder Variables
//I2C ports on multiplex board
int I2CA=1;
int I2CB=2;
int I2CC=3;


float newradA;
float newradB;
float newradC;

float initA;
float initB;
float initC;

float dradA=0,dradB=0,dradC=0;
float drevA=0,drevB=0,drevC=0;

////////////////////////////////////////
//////IMU
bool sensorCalibration = 0;
const int MPU_addr1 = 0x68;
float xa, ya, za, roll, pitch;
////// IMU
////////////////////////////////////////


void setup() {
    Serial.begin(9600); // For printing
    Wire.begin(); // Begin i2c

    EASYCAT.BufferIn.Cust.mpuCalibrated = sensorCalibration;

    ////////////////////////////////////
    ///IMU
    Wire.beginTransmission(MPU_addr1);                  //begin, send ...
       the slave adress (in this case 68)
    Wire.write(0x6B);                                   //make the ...
       reset (place a 0 into the 6B register)
    Wire.write(0);
    Wire.endTransmission(true);                         //end the ...
       transmission
    ////IMU
    ////////////////////////////////////////////



    ////////////////////////////////////////////////
    ///////// EVERYTHING FROM SETUP TOF
    // wait until serial port opens for native USB devices
    while (! Serial) {
        delay(1);
    }
```

```
Serial.println("Adafruit VL53L0X test.");
if (!lox.begin()) {
    Serial.println(F("Failed to boot VL53L0X"));
    while(1);
}
// power
Serial.println(F("VL53L0X API Continuous Ranging example\n\n"));

// start continuous ranging
lox.startRangeContinuous();
////////////////////// EVERYTHING FROM SETUP TOF
////////////////////////////////////////////////////////////


//////////////////////////
//////////////////////////
//Encoders//////
//Encoder1:
Wire.begin();
TCA9548(I2CA);
stpA.begin(4);
stpA.setDirection(AS5600_CLOCK_WISE);
int b = stpA.isConnected();
initA=stpA.getCumulativePosition()*AS5600_RAW_TO_RADIANS;
radA=0;
delay(20);

//Encoder2:
TCA9548(I2CB);
stpB.begin(4);
stpB.setDirection(AS5600_CLOCK_WISE);
int c = stpB.isConnected();
initB=stpB.getCumulativePosition()*AS5600_RAW_TO_RADIANS;
radB=0;
delay(20);

//Encoder3:
TCA9548(I2CC);
stpC.begin(4);
stpC.setDirection(AS5600_CLOCK_WISE);
int d = stpC.isConnected();
initC=stpC.getCumulativePosition()*AS5600_RAW_TO_RADIANS;
radC=0;
delay(20);
delay(1000);



// THIS NEEDS TO BE HERE TO ENABLE COMMUNICATION. Copied from IO ...
   script
if (EASYCAT.Init() == true)             // Initialization
{                                          // succesfully completed
    Serial.print ("initialized");         //
}                                            //

else                                        // Initialization failed
{                                 // The EasyCAT board was not ...
   recognized
    Serial.print ("initialization failed");       //
    // The most common reason is that the SPI
```

```
            // chip select choosen on the board doesn't
            // match the one choosen by the firmware

            // Stay in loop for ever
            // with the red led blinking fast
            while(1)                                                    //
            {                                                          //
                digitalWrite (13, LOW);                                //
                delay(100);                                              //
                digitalWrite (13, HIGH);                    //
                delay(100);                                                //
            }                                                          //
        }
        PreviousMillis = millis();
    }




    void loop() {
        // put your main code here, to run repeatedly:
        EcatState = EASYCAT.MainTask();
        Application();
        DiagPrint();
    }



    void Application(){
        encoders();
        IMUWithSIMPLE();
        TOFSensor();

        EASYCAT.BufferIn.Cust.encoder1Rad = radA;
        EASYCAT.BufferIn.Cust.encoder2Rad = radB;
        EASYCAT.BufferIn.Cust.encoder3Rad = radC;
        EASYCAT.BufferIn.Cust.xRotDeg = xRot;
        EASYCAT.BufferIn.Cust.yRotDeg = yRot;
        EASYCAT.BufferIn.Cust.TelescopeDistance_mm = TelescopeDistance_mm;
        EASYCAT.BufferIn.Cust.mpuCalibrated = sensorCalibration;
    }


    void IMUWithSIMPLE(){
        Wire.beginTransmission(MPU_addr1);
        Wire.write(0x3B);  //send starting register address, accelerometer ...
           high byte
        Wire.endTransmission(false); //restart for read
        Wire.requestFrom(MPU_addr1, 6); //get six bytes accelerometer data
        int t = Wire.read();
        xa = (t << 8) | Wire.read();
        t = Wire.read();
        ya = (t << 8) | Wire.read();
        t = Wire.read();
        za = (t << 8) | Wire.read();

        // formula from
//https://wiki.dfrobot.com/How_to_Use_a_Three-Axis_Accelerometer_for_Tilt_Sensing
        roll = atan2(ya , za) * 180.0 / PI;
```

```
    pitch = atan2(-xa , sqrt(ya * ya + za * za)) * 180.0 / PI; ...
       //account for roll already applied
    xRot = roll;
    yRot = Floatfilter2.add(pitch); // AVARAGE FILTER

}

void TOFSensor(){
    if (lox.isRangeComplete()) {
        float unFiltered = lox.readRange();

        // MOVING AVARAGE FILTER
        float AvarageFiltered = Floatfilter.add(unFiltered);
        TelescopeDistance_mm = AvarageFiltered;
    }
}



void encoders(){
    static uint32_t lastTime = 0;

    if (millis() - lastTime >= 5)
    {
        TCA9548(I2CA);
        newradA = ...
           stpA.getCumulativePosition()*AS5600_RAW_TO_RADIANS-initA;
        dradA=newradA-radA;
        radA=newradA;

        TCA9548(I2CB);
        newradB=stpB.getCumulativePosition()*AS5600_RAW_TO_RADIANS-initB;
        dradB=newradB-radB;
        radB=newradB;

        TCA9548(I2CC);
        newradC=stpC.getCumulativePosition()*AS5600_RAW_TO_RADIANS-initC;
        dradC = newradC-radC;
        radC=newradC;
    }
}


void TCA9548(int bus)
{
    Wire.beginTransmission(0x70);
    Wire.write(1 << bus);
    Wire.endTransmission();
}


void DiagPrint(){
    Serial.print("EncoderValue: ");
    Serial.print(radA);
    Serial.print(radB);
    Serial.print(radC);
    Serial.print(TelescopeDistance_mm);
    Serial.print(xRot);
    Serial.print(yRot);
    Serial.println();
}
```

B.2 OUTPUT_SLAVES

```
#include <Adafruit_PWMServoDriver.h>;
#define CUSTOM // Define custom variable is used
#include "OUTPUT_SLAVE.h"



#include "EasyCAT.h"                // EasyCAT library
#include <SPI.h>       // SPI library - The EasyCAT shield is ...
   connected to the SPI bus
EasyCAT EASYCAT;                    // EasyCAT istantiation
#include <Wire.h>                   // I2C library
//Global Copied from IO Script EasyCat
unsigned long Millis, PreviousMillis = 0;
unsigned char EcatState;
unsigned char N_Cycles = 0;
bool Blink = true;
UWORD Raw;



Adafruit_PWMServoDriver pwm1 = Adafruit_PWMServoDriver(0x40);

//INITIATES  WITH SERVO IN MIDDLE
int ServoPulseLength1 = 350;
int ServoPulseLength2 = 350;
int ServoPulseLength3 = 350;
int ServoPulseLength4 = 350;
int ServoPulseLength5 = 350;
int ServoPulseLength6 = 350;


// Servo Variables
int servoMin = 150; //(of 4096)  150 og 600 gives around 180 deg.
int servoMax = 600; //(of 4096)
int servoFrequency = 50; //Hz




void setup() {
    Serial.begin(9600); // For printing
    Wire.begin(); // Begin i2c

    // Servo
    pwm1.begin();
    pwm1.setOscillatorFrequency(27000000);
    pwm1.setPWMFreq(servoFrequency);


    // THIS NEEDS TO BE HERE TO ENABLE COMMUNICATION. Copied from IO ...
       script
    if (EASYCAT.Init() == true)                        // Initialization
    {                                                 // succesfully completed
        Serial.print ("initialized");                  //
    }                                                                 //

    else                                                  // ...
       Initialization failed
```

```
    {                                   // The EasyCAT board was not ...
       recognized
        Serial.print ("initialization failed");                         //
        // The most common reason is that the SPI
        // chip select choosen on the board doesn't
        // match the one choosen by the firmware

        // Stay in loop for ever
        // with the red led blinking fast
        while(1)                                                        //
        {                                                               //
            digitalWrite (13, LOW);                                   //
            delay(100); ...
                                                                    //
            digitalWrite (13, HIGH);                                  //
            delay(100); ...
                                                                    //
        }                                                               //
    }
    PreviousMillis = millis();
}



void loop() {
    // put your main code here, to run repeatedly:
    EcatState = EASYCAT.MainTask();
    Application();
    DiagPrint();
}

//
float servoSafety(int ServoPulseLength){
    int  servoPulse;
    if (ServoPulseLength<= 250 & ServoPulseLength != 0)
    {
        servoPulse = 250;
    }
    else if (ServoPulseLength>= 450)
    {
        servoPulse = 450;}
    else if (ServoPulseLength== 0)
    {
        servoPulse = 350;}
    else
    {
        servoPulse = ServoPulseLength;
    }
    return servoPulse;
}


void Application(){
    ServoPulseLength1 = EASYCAT.BufferOut.Cust.ServoPulseLength1;
    ServoPulseLength2 = EASYCAT.BufferOut.Cust.ServoPulseLength2;
    ServoPulseLength3 = EASYCAT.BufferOut.Cust.ServoPulseLength3;
    ServoPulseLength4 = EASYCAT.BufferOut.Cust.ServoPulseLength4;
    ServoPulseLength5 = EASYCAT.BufferOut.Cust.ServoPulseLength5;
    ServoPulseLength6 = EASYCAT.BufferOut.Cust.ServoPulseLength6;
```

```
        ServoPulseLength1 = servoSafety(ServoPulseLength1);
        ServoPulseLength2 = servoSafety(ServoPulseLength2);
        ServoPulseLength3 = servoSafety(ServoPulseLength3);
        ServoPulseLength4 = servoSafety(ServoPulseLength4);
        ServoPulseLength5 = servoSafety(ServoPulseLength5);
        ServoPulseLength6 = servoSafety(ServoPulseLength6);

        pwm1.setPWM(1, 0, ServoPulseLength1);
        pwm1.setPWM(2, 0, ServoPulseLength2);
        pwm1.setPWM(3, 0, ServoPulseLength3);
        pwm1.setPWM(4, 0, ServoPulseLength4);
        pwm1.setPWM(5, 0, ServoPulseLength5);
        pwm1.setPWM(6, 0, ServoPulseLength6);
    }



    void DiagPrint(){
        Serial.print("     ServoValue:    ");
        Serial.print(ServoPulseLength1);
        Serial.print(ServoPulseLength2);
        Serial.print(ServoPulseLength3);
        Serial.print(ServoPulseLength4);
        Serial.print(ServoPulseLength5);
        Serial.println(ServoPulseLength6);

    }
```

B.3 theta_i_2_thetha_k()

```
function [theta_k,xk] = theta_i_2_thetha_k(theta_i)
%Constant Leght all mm:
lh = 116.62;
li = 121.93;
lj = 13.34;
lk = 30;
lm = 20;
ln = 16;
theta_h = acos((lh^2+lj^2-li^2)/(2*lh*lj));

%Based on input:
%Cylinder lenght:
xk = sqrt(lh^2+lk^2-2*lh*lk*cos(theta_i));

theta_j = theta_h-theta_i;
ll = sqrt(lj^2+lk^2-2*lj*lk*cos(theta_j));

theta_n = acos((ll^2+ln^2-lm^2)/(2*ll*ln));
theta_l = acos((li^2+ll^2-xk^2)/(2*li*ll));
theta_k = theta_n + theta_l;
end
```

B.4 theta_i_2_thetha_k()

```
function q3  = theta_i_2_thetha_k(theta_i)
%Validated with physical measurements on crane from 0 - 180 deg

%Constant Leght all mm:

theta_i_init = 2.3527;%start angle when piston is fully extracted
theta_k_init = 3.9706; %start angle when piston is fully extracted
Theta_I = theta_i_init + theta_i;

lh = 116.62;
li = 121.93;
lj = 13.34;
lk = 30;
lm = 20;
ln = 16;
theta_h = real(acos((lh^2+lj^2-li^2)/(2*lh*lj)));

%Based on input:
%Cylinder lenght:
xk = sqrt(lh^2+lk^2-2*lh*lk*cos(Theta_I));

theta_j = theta_h-Theta_I;
ll = sqrt(lj^2+lk^2-2*lj*lk*cos(theta_j));

theta_n = real(acos((ll^2+ln^2-lm^2)/(2*ll*ln)));
theta_l = acos((li^2+ll^2-xk^2)/(2*li*ll));
theta_k = theta_n + theta_l;
q3 = theta_k-theta_k_init; % - init so output start at 0
end
```

B.5 forwardKinematics()

```
    function [x,y,z] = forwardKinematics(q1, q2, q3, q4)

    %Lenght from assy drawing
    l1 = 156.9/1000;
    l2 = 27.45/1000;
    l3 = 179.4/1000;
    l4 = 26.86/1000;

    l5 = 167.25/1000;
    l6 = 1.75/1000; %m
    W1 = -l2+l3+l5;
    W2 = l4;
    W3 = l1-l6;


    % Position of each joint
    a1 = [0; 0; 0];
    a2 = [-l2; 0; l1];
    a3 = [-l2+l3; 0; l1];
    a4 = [W1; W2; W3];
    a5 = [W1; W2; W3];
    a6 = [W1; W2; W3];

    % Screw axis of ech joint
    Sw1 = [0; 0; 1];
```

```
Sw2 = [0; 1; 0];
Sw3 = [0; 1; 0];
Sw4 = [0; 0; 0];
Sw5 = [0; 1; 0];
Sw6 = [0; 0; 1];

Sv1 = cross(a1,Sw1);
Sv2 = cross(a2,Sw2);
Sv3 = cross(a3,Sw3);
Sv4 = [1;0;0];%translation along xaxis
Sv5 = cross(a5,Sw5);
Sv6 = cross(a6,Sw6);

S1 = [Sw1;Sv1];
S2 = [Sw2;Sv2];
S3 = [Sw3;Sv3];
S4 = [Sw4;Sv4];
S5 = [Sw5;Sv5];
S6 = [Sw6;Sv6];

M = [[1 0 0 W1];
[0 1 0 W2];
[0 0 1 W3];
[0 0 0 1];];


% Joints:

q5 = 0;
q6 = 0;

thetalist = [q1; q2 ; q3; q4; q5; q6];
Slist = [S1,S2,S3,S4,S5,S6];



T = M;

for i = 6: -1: 1 % 6 = number of joints

T = MatrixExp6(VecTose3(Slist(:, i) * thetalist(i))) * T;
end
x = T(1,4);
y = T(2,4);
z = T(3,4);
end
```

B.6 inverseVelocityKinematics()

```
function [q1Dot,q2Dot,q3Dot,q4Dot] = ...
    inverseVelocityKinematics(q1,q2,q3,q4,Vx,Vy,Vz)

Vs = [0;0;0;Vx;Vy;Vz];

q5 = -(q2+q3); % ALWAYS HORISONTAL DUMMY JOINT
q6 = -q1; % ALWAYS PARRALELL TO ORGIN
```

```
    Slist = [[0            0            0            0            0 ...
                  0];
    [0            1            1            0            1            0];
    [1            0            0            0            0            1];
    [0       -0.1569      -0.1569            1      -0.15515      0.02686];
    [0            0            0            0            0     -0.31922];
    [0       -0.02745      0.15197            0      0.31922            ...
       0]]; %m


    thetalist = [q1; q2 ; q3; q4; q5; q6];

    J = JacobianSpace(Slist, thetalist);

    thetalistDot = pinv(J)*Vs;
    q1Dot = thetalistDot(1);
    q2Dot = thetalistDot(2);
    q3Dot = thetalistDot(3);
    q4Dot = thetalistDot(4);
```

B.7 inverseKinematics()

```
function [q1, q2,q3,q4, success] = ...
   inverseKinematics(q1Inn,q2Inn,q3Inn,q4Inn,X,Y,Z)
q5Inn = -(q2Inn+q3Inn); % ALWAYS HORISONTAL DUMMY JOINT
q6Inn = -q1Inn; % ALWAYS PARRALELL TO ORGIN

%Lenght from assy drawing
l1 = 156.9/1000;%m
l2 = 27.45/1000;%m
l3 = 179.42/1000;%m
l4 = 26.86/1000;%m
l5 = 167.25/1000; %m
l6 = 1.75/1000; %m

W1 = -l2+l3+l5;
W2 = l4;
W3 = l1-l6;

% Position of each joint
a1 = [0; 0; 0];
a2 = [-l2; 0; l1];
a3 = [-l2+l3; 0; l1];
a5 = [W1; W2; W3];
a6 = [W1; W2; W3];

% Screw axis of ech joint
Sw1 = [0; 0; 1];
Sw2 = [0; 1; 0];
Sw3 = [0; 1; 0];
Sw5 = [0; 1; 0];
Sw6 = [0; 0; 1];

Sv1 = cross(a1,Sw1);
Sv2 = cross(a2,Sw2);
Sv3 = cross(a3,Sw3);
Sv5 = cross(a5,Sw5);
Sv6 = cross(a6,Sw6);
```

```
S1 = [Sw1;Sv1];
S2 = [Sw2;Sv2];
S3 = [Sw3;Sv3];
S4 = [0;0;0;1;0;0];
S5 = [Sw5;Sv5];
S6 = [Sw6;Sv6];

M = [[1 0 0 W1];
[0 1 0 W2];
[0 0 1 W3];
[0 0 0 1];];

Slist = [S1,S2,S3,S4,S5,S6];


TInverseDesired = [[1, 0, 0, X]; [0, 1, 0, Y]; [0, 0, 1, Z]; [0, 0, 0, 1]];
thetalist0 = [q1Inn; q2Inn; q3Inn;q4Inn; q5Inn; q6Inn];
eomg = 0.00001;
ev = 0.00001;
[thetalist, success] = IKinSpace(Slist, M, TInverseDesired, thetalist0, ...
   eomg, ev);


q1 = thetalist(1);
q2 = thetalist(2);
q3 = thetalist(3);
q4 = thetalist(4);
```

B.8 inverseCombinedKinematics()

```
    function [q1, q2,q3,q4Dot, success] = ...
       inverseCombinedKinematics(q1Inn,q2Inn,q3Inn,q4Inn,X,Y,Z,Vx,Vy,Vz)

    q5Inn = -(q2Inn+q3Inn); % ALWAYS HORISONTAL DUMMY JOINT
    q6Inn = -q1Inn; % ALWAYS PARRALELL TO ORGIN

    %Lenght from assy drawing
    l1 = 156.9/1000;%m
    l2 = 27.45/1000;%m
    l3 = 179.42/1000;%m
    l4 = 26.86/1000;%m

    %From Telescope drawing (IN assy drawing)
    l5 = 167.25/1000; %%m Corrected after sensor calibration
    l6 = 1.75/1000; %m
    W1 = -l2+l3+l5+q4Inn; % NOTE: q4 lenght added here, because 3 DOF ...
       using current lenght
    W2 = l4;
    W3 = l1-l6;


    % Position of each joint
    a1 = [0; 0; 0];
    a2 = [-l2; 0; l1];
    a3 = [-l2+l3; 0; l1];
    a5 = [W1; W2; W3];
    a6 = [W1; W2; W3];
```

```
% Screw axis of ech joint
Sw1 = [0; 0; 1];
Sw2 = [0; 1; 0];
Sw3 = [0; 1; 0];
Sw5 = [0; 1; 0];
Sw6 = [0; 0; 1];

Sv1 = cross(a1,Sw1);
Sv2 = cross(a2,Sw2);
Sv3 = cross(a3,Sw3);
Sv5 = cross(a5,Sw5);
Sv6 = cross(a6,Sw6);

S1 = [Sw1;Sv1];
S2 = [Sw2;Sv2];
S3 = [Sw3;Sv3];
S4 = [0;0;0;1;0;0];
S5 = [Sw5;Sv5];
S6 = [Sw6;Sv6];

M = [[1 0 0 W1];
[0 1 0 W2];
[0 0 1 W3];
[0 0 0 1];];

Slist = [S1,S2,S3,S5,S6];


TInverseDesired = [[1, 0, 0, X]; [0, 1, 0, Y]; [0, 0, 1, Z]; [0, 0, 0, ...
    1]];
thetalist0 = [q1Inn; q2Inn; q3Inn; q5Inn; q6Inn];
eomg = 0.00001;
ev = 0.0000001;
[thetalist, success] = IKinSpace(Slist, M, TInverseDesired, ...
    thetalist0, eomg, ev);


%For velocity Kinematics 4 dof.
Vs = [0; 0;0; Vx; Vy; Vz];
thetalist0_4DOF = [q1Inn; q2Inn; q3Inn; q4Inn; q5Inn; q6Inn];
Slist_4DOF = [S1,S2,S3,S4,S5,S6];
J = JacobianSpace(Slist_4DOF, thetalist0_4DOF);
thetalistDot = pinv(J) * Vs;

q1 = thetalist(1);
q2 = thetalist(2);
q3 = thetalist(3);
q4Dot = thetalistDot(4);
```

B.9 Find transformation matrix using DH

```
clear; close all; clc;


% Joints:
q1 = 0;
q2 = 0;
```

```
q3 = 0;
q4 = 0;


%Lenght from assy drawing
l1 = 156.9;
l2 = 27.45;
l3 = 179.42;
l4 = 26.86;
%From Telescope drawing (IN assy drawing)
l5 = 167.25; % Corrected after sensor calibration
l6 = 1.75; %mm


% Update for every link:
theta_i = q1;
d_i = l1;
a_i = -l2;
alpha_i = -pi/2;

T1 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];


theta_i = q2;
d_i = 0;
a_i = l3;
alpha_i = 0;


T2 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];


theta_i = q3-pi/2;
d_i = l4;
a_i = 0;
alpha_i = -pi/2;


T3 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];

theta_i = 0;
d_i = l5+q4;
a_i = -l6;
alpha_i = 0;
```

```
T4 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];

T2Pos = T1;
T3Pos = T1*T2;
T4Pos = T1*T2*T3;
TendEffector = T1*T2*T3*T4

X1= [0 ,0 , 0];
X2 = [T2Pos(1,4),T2Pos(2,4),T2Pos(3,4)];
X3 = [T3Pos(1,4),T3Pos(2,4),T3Pos(3,4)];
X4 = [T4Pos(1,4),T4Pos(2,4),T4Pos(3,4)];
X5 = [TendEffector(1,4),TendEffector(2,4),TendEffector(3,4)];

xyz = vertcat(X1,X2,X3,X4,X5);
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'k--')
hold on
xlim([-40 400])
ylim([-200 200])
zlim([0 400])
grid on


% xTest = -l2*cos(q1) + l3*cos(q2)-l6
% yTest = -l2*sin(q1) + l3*sin(q2)
% zTest = l1 + l4 + l5 + q4

%%

% NEXT POSITION %%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

% Joints:
q1 = -pi/10;
q2 = -pi/4;
q3 = pi/2;
q4 = 100;


%Lenght from assy drawing
l1 = 156.9;
l2 = 27.45;
l3 = 179.42;
l4 = 26.86;
%From Telescope drawing (IN assy drawing)
%l5 = 163.43;
l5 = 167.25; % Corrected after sensor calibration

% Update for every link:
theta_i = q1;
d_i = l1;
a_i = -l2;
alpha_i = -pi/2;

T1 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
```

```
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];


theta_i = q2;
d_i = 0;
a_i = l3;
alpha_i = 0;


T2 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];


theta_i = q3-pi/2;
d_i = l4;
a_i = 0;
alpha_i = -pi/2;


T3 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];

theta_i = 0;
d_i = l5+q4;
a_i = 0;
alpha_i = 0;


T4 = [cos(theta_i) -sin(theta_i)*cos(alpha_i) ...
    sin(theta_i)*sin(alpha_i) a_i*cos(theta_i);
sin(theta_i) cos(theta_i)*cos(alpha_i) -cos(theta_i)*sin(alpha_i) ...
    a_i*sin(theta_i);
0 sin(alpha_i) cos(alpha_i) d_i;
0 0 0 1];

T2Pos = T1;
T3Pos = T1*T2;
T4Pos = T1*T2*T3;
TendEffector = T1*T2*T3*T4

X1= [0 ,0 , 0];
X2 = [T2Pos(1,4),T2Pos(2,4),T2Pos(3,4)];
X3 = [T3Pos(1,4),T3Pos(2,4),T3Pos(3,4)];
X4 = [T4Pos(1,4),T4Pos(2,4),T4Pos(3,4)];
X5 = [TendEffector(1,4),TendEffector(2,4),TendEffector(3,4)];

xyz = vertcat(X1,X2,X3,X4,X5);
plot3(xyz(:,1),xyz(:,2),xyz(:,3),'k-')
hold on
xlim([-40 400])
```

```
    ylim([-200 200])
    zlim([0 400])
    grid on
```

Appendix C

Appendix Figures

C.1 2D Hand Drawing

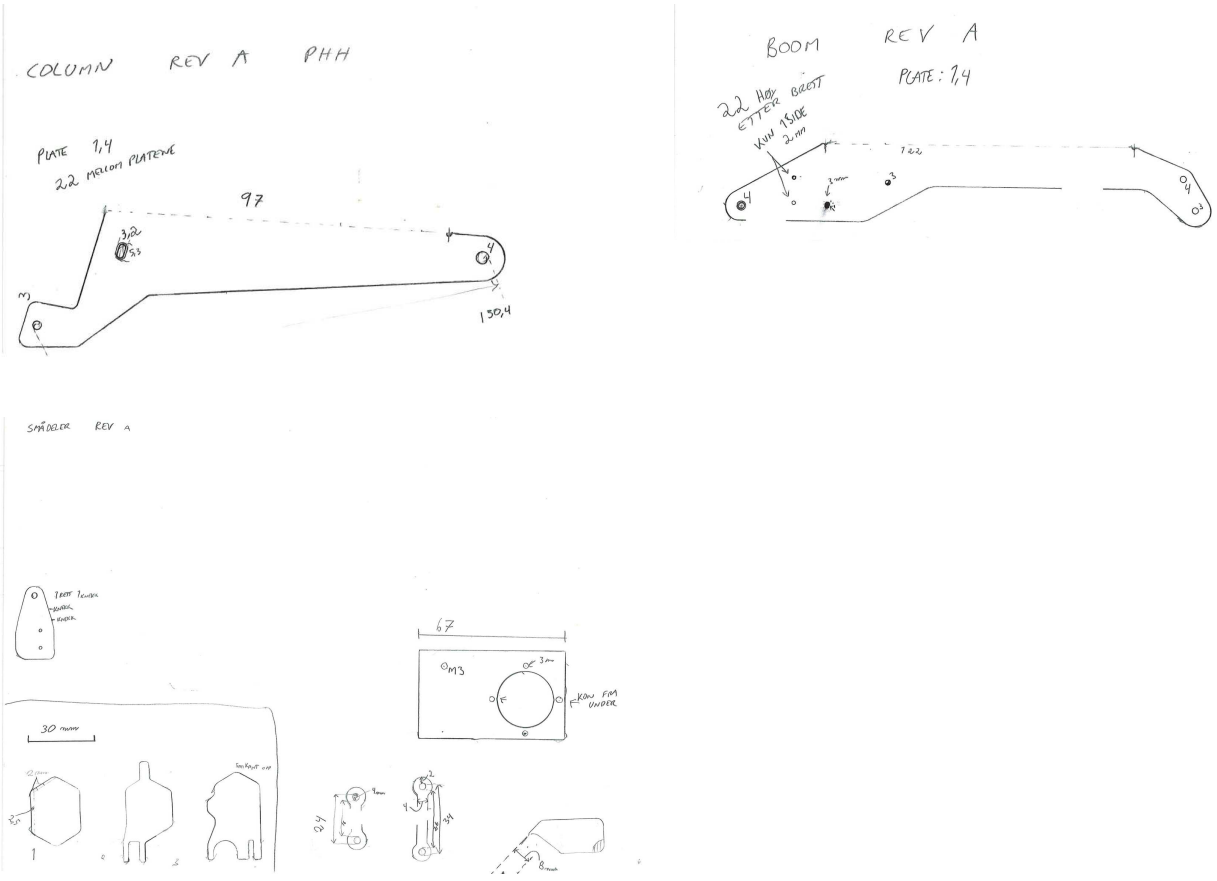

Figure C.1: Hand Drawing

C.2 Class Diagram PLC Program


<<program>>
P_VelToPos
fbToolpointPositionIntegrator FB_ToolpointPositionIntegrator
<<program>>
P_ManualControl
<<program>>
P_Idle
<<program>>
P_CartesianController
fbPIDController FB_PID_Controller
fbInverseKinematics FB_InverseKinematics
<<program>>
P_CarteseanControllerServoActuation4DOF
<<program>>
P_SensorScaling
fbEncoderval2Angleq3 FB_Encoderval2Angle_q3
fbDistSensorValue2Distance_q4 FB_DistSensorValue2Distance_q4
tempTOF1 LREAL
tempTOF11 LREAL
tempCounter DINT
<<program>>
P_SlaveInputControl
tempEncoder1 LREAL
tempEncoder11 LREAL
<<program>>
P_ModeControl
<<program>>
P_NormalizeJointVel
<<program>>
P_Outputs
<<program>>
P_PathGeneration
fbPathGenerator FB_PathGenerator
Q1 BOOL
ElapsedTime LREAL
TimeX LREAL
TimeY LREAL
TimeZ LREAL
TimeEachSequence INT
cartesianVelocity LREAL
<<program>>
MAIN
fbDistSensorValue2Distance_q4
FB_DistSensorValue2Distance_q4
DistSensorVal REAL
DistSensorMin REAL
DistSensorMax REAL
q4Min REAL
q4Max REAL
fSensorValue REAL {input}
f_q4 REAL {output}
<<program>>
P_Kinematics
fbForwardKinematics FB_ForwardKinematics
<<program>>
P_SetZero
<<program>>
P_Heave
fbGet3r FB_Get3r
fbGenr1b FB_Genr1b
fbToolpointPositionIntegrator2 FB_ToolpointPositionIntegrator
thethaThreshold LREAL
<<program>>
P_JoystickInput
fbJoystickLeftVert FB_JoystickInputScaling
fbJoystickLeftHor FB_JoystickInputScaling
fbJoystickRightVert FB_JoystickInputScaling
fbJoystickRightHor FB_JoystickInputScaling
bEnableJoystick BOOL
<<program>>
P_CartesianVelFromJoystick
fbJoystickToCartesianVelocity1 FB_JoystickToCartesianVelocity
fbJoystickToCartesianVelocity2 FB_JoystickToCartesianVelocity
fbJoystickToCartesianVelocity3 FB_JoystickToCartesianVelocity
<<program>>
P_ActuatorLimits
fbActuatorLimitsq1 FB_ActuatorLimits
fbActuatorLimitsq2 FB_ActuatorLimits
fbActuatorLimitsq3 FB_ActuatorLimits
fbActuatorLimitsq4 FB_ActuatorLimits
<<program>>
P_CartesianLimit
<<function>>
NormToPulseLenght
<<function>>
ButtonControlPulseLenght
fActuation LREAL
fServoPulseLenght LREAL
fJoystickButtonFront BOOL {input}
fJoystickButtonBack BOOL {input}
fNeutralPosition LREAL {input}
fButtonControlPulseLenght UINT {readonly,output}
fbEncoderval2Angleq3
fbActuatorVelToNormalized4
FB_ActuatorVelFeedForward
fActuatorVelocity LREAL {input}
fMaxActuatorVelocity LREAL {input}
fVelocityFeedForward LREAL {output}
fbInverseKinematics
FB_InverseKinematics
fbPathGenerator
FB_PathGenerator
tTimer TON
tElapsed TIME
bStart BOOL {input}
TIN TIME {input}
Qout BOOL {output}
t LREAL {output}
fbToolpointPositionIntegrator
fbToolpointPositionIntegrator2
FB_ToolpointPositionIntegrator
cycleTime LREAL
fXref LREAL {input}
fYref LREAL {input}
fZref LREAL {input}
fXrefOut LREAL {output}
fYrefOut LREAL {output}
fZrefOut LREAL {output}
FB_Encoderval2Angle_q3
fbJoystickToCartesianVelocity3
fbJoystickToCartesianVelocity2
fbJoystickToCartesianVelocity1
FB_JoystickToCartesianVelocity
fNormalizedJoistick LREAL {input}
fScalingFactor LREAL {input}
fAxisVelocity LREAL {output}
fbForwardKinematics
FB_ForwardKinematics
fbPIDController
FB_PID
Error REAL
Integral REAL
LastError REAL
Derivative REAL
timestep REAL
SetPoint LREAL {input}
SensorValue LREAL {input}
Kp LREAL {input}
Ki LREAL {input}
Kd LREAL {input}
MaxOutput LREAL {input}
MinOutput LREAL {input}
ResetIntegral BOOL {input}
Output LREAL {output}
FB_PID_Controller
Kp1 LREAL
Kp2 LREAL
Kp3 LREAL
Ki1 LREAL
Ki2 LREAL
Ki3 LREAL
Kd1 LREAL
Kd2 LREAL
Kd3 LREAL
KF4 LREAL
FFPart4 LREAL
FFPart4_B LREAL
pid1 FB_PID
pid2 FB_PID
pid3 FB_PID
fbActuatorLimitsq4
fbActuatorLimitsq3
fbActuatorLimitsq2
fbActuatorLimitsq1
FB_CartesianVelLimit
axisVelocityIn LREAL {input}
axisPosition LREAL {input}
axisMinPosition LREAL {input}
axisMaxPostion LREAL {input}
axisVelocityOut LREAL {output}
FB_ActuatorLimits
fsensorMinValue LREAL {input}
fsensorMaxValue LREAL {input}
fSensorSignal LREAL {input}
fServoSignalIn LREAL {input}
bMinReached BOOL {output}
bMaxReached BOOL {output}
fServoSignalOut LREAL {output}
fbJoystickRightHor
fbJoystickRightVert
fbJoystickLeftHor
fbJoystickLeftVert
FB_JoystickInputScaling
stJoystickInputScaling
<<struct>>
ST_Parameters
fCyclicTime LREAL
stJoystickInputScalingLeftX ST_JoystickInputScaling
stJoystickInputScalingLeftY ST_JoystickInputScaling
stJoystickInputScalingRightX ST_JoystickInputScaling
stJoystickInputScalingRightY ST_JoystickInputScaling
<<struct>>
ST_JoystickInputScaling
fMaxInputValue LREAL
fMinInputValue LREAL
fNeutralInputValue LREAL
fNeutralDeadband LREAL
fMaxOutputValue LREAL
fMinOutputValue LREAL
fNeutralOutputValue LREAL
stStatus
<<struct>>
ST_AnalogInputStatus
Underrange BIT
Overrange BIT
Limit_1_0 BIT
Limit_1_1 BIT
Limit_2_0 BIT
Limit_2_1 BIT
Error BIT
Reserved_0_7 BIT
Reserved_1_0 BIT
Reserved_1_1 BIT
Reserved_1_2 BIT
Reserved_1_3 BIT
Reserved_1_4 BIT
Reserved_1_5 BIT
TxPDO_State BIT
TxPDO_Toggle BIT
FB_AnalogInputScaling
fRawInput LREAL
fNormalizedInput LREAL
nRawInput INT {input}
stAnalogInputScaling ST_AnalogInputScaling {input}
fOutput LREAL {output}
stAnalogInputScaling
<<struct>>
ST_AnalogInputScaling
fMaxInputValue LREAL
fMinInputValue LREAL
fMaxOutputValue LREAL
fMinOutputValue LREAL
stParam
<<struct>>
ST_AnalogInput
stStatus ST_AnalogInputStatus
nValue INT
stJoyRX
stJoyLY
stJoyRX
stJoyRY
stJoyLX
stJoyLY
<<global>>
G_InverseKinematics
fvXref LREAL
fvYref LREAL
fvZref LREAL
fq4Dotref LREAL
f_q1_ref LREAL
f_q2_ref LREAL
f_q3_ref LREAL
bKinematicsSucsessful BOOL
<<global>>
G_SensorsScaled
q1 REAL
q2 REAL
q3 REAL
q4 REAL
q1Dot REAL
q2Dot REAL
q3Dot REAL
q4Dot REAL
<<global>>
G_SensorInput
fEncoder1Rad REAL
fEncoder2Rad REAL
fEncoder3Rad REAL
fTelescopeDistance_mm REAL
fXRotDeg REAL
fYRotDeg REAL
bSensorDataLost BOOL
<<global>>
G_Path
bPathRunning BOOL
bPathFinnish BOOL
bStartPath BOOL
tTime LREAL
bPathActivated BOOL
<<global>>
G_Toolpoint
fX LREAL
fY LREAL
fZ LREAL
fXref LREAL
fYref LREAL
fZref LREAL
<<global>>
G_ScalingParameters
stParam ST_Parameters
<<global>>
G_Outputs
bGreenButtonLight BOOL
bRedButtonLight BOOL
bSafetyRelayReset BOOL
<<global>>
G_ActuatorLimits
fq1Min LREAL
fq1Max LREAL
fq2Min LREAL
fq2Max LREAL
fq3Min LREAL
fq3Max LREAL
fq4Min LREAL
fq4Max LREAL
bq1Min BOOL
bq1Max BOOL
bq2Min BOOL
bq2Max BOOL
bq3Min BOOL
bq3Max BOOL
bq4Min BOOL
bq4Max BOOL
fZMin LREAL
fZMax LREAL
fYMin LREAL
fYMax LREAL
fXMax LREAL
fXMin LREAL
<<global>>
G_Joystick
fJoystickLeftNormalizedY LREAL
fJoystickLeftNormalizedX LREAL
fJoystickRightNormalizedY LREAL
fJoystickRightNormalizedX LREAL
nJoystickLeftDirectionX DINT
nJoystickLeftDirectionY DINT
nJoystickRightDirectionX DINT
nJoystickRightDirectionY DINT
bJoystick1InUse BOOL
bJoystick2InUse BOOL
bJoystick3InUse BOOL
bJoystick4InUse BOOL
<<global>>
G_ControlBoxInputs
bManualModeSelector BOOL
bAutoModeSelector BOOL
bRedButton BOOL
bGreenButton BOOL
bJoystickLeftEnable BOOL
bJoystickRightEnable BOOL
bSafetyRelayRestartButton BOOL
stJoyLX ST_AnalogInput
stJoyLY ST_AnalogInput
stJoyRX ST_AnalogInput
stJoyRY ST_AnalogInput
<<global>>
G_Buttons
<<global>>
G_Heave
fThetaStart LREAL
fTheta LREAL
r3x LREAL
r3y LREAL
r3z LREAL
r1bxCrane LREAL
r1byCrane LREAL
r1bzCrane LREAL
bHeaveEnable BOOL
bHeaveActive BOOL
<<global>>
G_Servo
nServoPulseLenght1 UINT
nServoPulseLenght2 UINT
nServoPulseLenght3 UINT
nServoPulseLenght4 UINT
nServoPulseLenght5 UINT
nServoPulseLenght6 UINT
fNormalizedActuatorVelFeedForward_q1Dot LREAL
fNormalizedActuatorVelFeedForward_q2Dot LREAL
fNormalizedActuatorVelFeedForward_q3Dot LREAL
fNormalizedActuatorVelFeedForward_q4Dot LREAL
q1_position_error_part LREAL
q2_position_error_part LREAL
q3_position_error_part LREAL
q4_position_error_part LREAL

C.3 Cover Panels For Demo Model