UNIVERSITY OF AGDER

Simulation and Control of an Anti-Swing System for a Suspended Load attached to a moving Base Robot

by

Jan Thomas G. Olsen

Supervisor

Sondre Sanden Tørdal

This master's thesis is carried out as a part of the education at the University of Agder and is therefore approved as a part of this education. However, this does not imply that the University answers for the methods that are used or the conclusions that are drawn.

University of Agder, 2018

Faculty of Engineering and Science

Department of Engineering Sciences

Abstract

Advanced motion compensation is an important field of engineering in today's offshore industry. Performing advanced load handling operations at sea requires high attention to both safety and efficiency. In environments governed by wave motion and harsh weather conditions, these types of operations are complex tasks. Most of the load handling scenarios are performed by advanced offshore loader cranes, where many of these are equipped with the industry's state-of-the-art Active Heave Compensation. This technology is capable of compensating for the wave height disturbance, meaning that the load is kept at a constant height above the seafloor. A common problem arises when equipment or personnel have to be transported from a moving vessel to another vessel or offshore installation. In these scenarios compensation of the side-to-side motion is of equal importance as the relative height movement.

This thesis proposes a method to reduce this side-ways motion. An anti-swing system has been developed for a simulation model of a suspended load attached to a moving base robot. The work focuses on deriving mathematical models of the related systems, where control systems are designed to the reduce the swing motion of the suspended load by actuation of the load handling robot. The developed system models are based on the available equipment of the Norwegian Motion-Laboratory. Results of the proposed system are obtained from the simulation of the motion system, which yields a system capable of tracking the robotic tool-point reference signal with acceptable accuracy and reducing the suspended load's swing-angles with satisfactory performance.

Preface

This master's thesis is written and conducted at the University of Agder, campus Grimstad. It utilizes the available configuration and equipment of the Norwegian Motion Laboratory at the University, and it is carried out with the aim of expanding the current extent of the facility, where the result can be used for further research and experimental work.

I wish to state my greatest gratitude towards my supervisor Sondre Sanden Tørdal, for the support and guidance I have received throughout this project, without whom none of this would be possible.

Jan Thomas G. Olsen

Grimstad, May 24, 2018

Contents

List of Figures

List of Tables

1 Introduction

1.1 Background and Motivation

Advanced motion compensation is a topic with increased interest in the offshore industry, especially when considering offshore crane load handling operations. Load operations involving floating wind turbines, autonomous shipping and vessel-to-vessel loading. are a few examples of scenarios where motion compensation is a subject for improved safety and performance. Active Heave Compensation (AHC) is a technology widely used by current load handling cranes [1]. These are capable of controlling the load's height while the vessel is experiencing a wave-induced motion, in 6 degrees of freedom (6-DOF). To safely and efficiently handle scenarios where a crane should transfer a hanging load from a moving vessel to a secondary vessel or offshore installation, an extension of the AHC technology is desired. 6-DOF wave motion will not only affect the height of a suspended load but also influence the swing motion of the load. Today such operations are limited by the weather window, where the significant wave height should be below 2.5 $[m]$ to allow for such load transfers [2]. Introducing an anti-swing system can reduce the load's pendulated movements caused by the relative wave motion of the vessel, i.e., implementing a control algorithm to compensate for the suspended load's surge and sway motion. Which can extend the window for safe load handling operation.

A recognized process of measuring the relative motion of an offshore vessel is using Motion Reference Units (MRU). Installing one or multiple MRUs, the wave motion experienced by the vessel can be obtained and used as a reference signal to a compensation system. Experimental research done by [3], utilizes this method of using MRUs to measure the important DOFs. Regarding the scenario of a suspended load attached to an offshore crane, a way of measuring the position of the load is necessary to implement an anti-swing system. Techniques of motion detection can be applied, with the use of laser trackers or vision systems. Motion tracking by using a vision system and Aruco markers is investigated by [4]. Motion tracking of a moving target was conducted by [5], where a laser tracker and a tracking probe was used to measure the relative motion between two wave-induced platforms.

The Norwegian Motion-Laboratory is a research facility located at the University of Agder. This laboratory consists of a couple of wave simulation platforms which, together with the available industrial robot and measurement systems, enables the possibility to conduct experiments and research related to offshore motion compensation and load handling operations.

1.2 Problem Description

In this master thesis, the primary objective is to design and develop a simulation model capable of compensating for the swing angles of a suspended load attached to a moving base robot. The system will inherit the functionality of the Norwegian Motion-Laboratory, where one of the motion platforms will be simulated to generate a wave motion. The industrial robot will be restricted to act as a 3-DOF offshore crane and will experience relative motion at its base. The motion of the robot base will influence the motion of the attached load, where the swing angle should be compensated for, by using the robot as an actuator.

A more specific description of the project objectives is presented as:

- Get familiar with the current infrastructure of the Norwegian Motion-Laboratory
- Kinematic analysis of the industrial robot should be performed, where a mathematical simulation model of the system should be developed.
- Analysis of the kinematics and dynamics of a suspended load connected to a moving attachment point should be conducted.
- Kinematic analysis of the full motion system should be identified.
- A combined full system simulation model should be developed, for the suspended load attached to the industrial robot, where the system is influenced by 6-DOF platform motion.
- An anti-swing control system capable of compensating for the load's swing motion should be designed and implemented to the simulation models.

1.3 Assumption and Limitations

1.3.1 Industrial Robot

Control of the industrial robot is considered to be achievable through actuation of the angular joint motion. The robot is assumed to act as a rigid system, where no dynamic analysis is necessary for the joint actuation. Feedback from the robot's tool-point position and velocity is considered to be available.

1.3.2 Suspended Load

The suspended load can be assumed to act as a 3-dimensional pendulum, where the wire is considered as a massless rigid rod. Elongation and deflection of the wire are not to be considered in this thesis, and the small frictional elements between the robot's tool-point and the wire are neglectable.

1.3.3 Motion Platform

Measurements for the relative motion of the platform are assumed to be time continuous and available for the simulation experiments.

1.4 Report Outline

A short description of the chapters and their contents are given by the following bullet points:

- **Chapter 1 - Introduction**
  A description of the motivation and background, together with the problem statement for this master thesis is presented.
- **Chapter 2 - Motion Laboratory**
  This chapter presents the Norwegian Motion Laboratory, a research facility located at the University of Agder. A brief introduction to the available equipment and the current configuration of the facility is presented.
- **Chapter 3 - Theory**
  In this chapter, the background theory and fundamental equations are introduced. This includes topics of the robot kinematics, analysis of the kinematics and dynamics of the suspended load. Combined motion system analysis, where the motion platform is connected with the industrial robot and suspended load system. A description of control system design, will also be introduced by this chapter.
- **Chapter 4 - Method**
  Here, the method and approach is presented for the modelling and simulation of the presented systems. With the use of Matlab and Simulink, mathematical models of the industrial robot, suspended load, and motion platform are made. This chapter also presents and describes the implementation of the different control systems.
- **Chapter 5 - Results**
  This chapter presents the reader with the results for the conducted simulations of the developed models and control systems.
- **Chapter 6 - Discussion**
  The reader will be presented with a discussion of the work conducted for this master thesis, together with suggestions for further work.
- **Chapter 7 - Conclusion**
  This chapter elaborates the author's main conclusions and findings for the related work.

1.4.1 GitHub Repository

Throughout this thesis, computer software such as Matlab and Simulink have been used to aid in the design and development of the mathematical system models and control system. The related scripts are provided by the appendices. Also, a GitHub repository has been constructed for this master thesis. Here, it is possible to download the scripts as well as the full simulation models for the system models and control systems.

`https://github.com/jantolsen/motionlab-anti-swing-system`.

2 Motion Laboratory

The Norwegian Motion Laboratory (Motion-Lab) [6] is a research establishment in the Mechatronics Laboratory located at the University of Agder (UiA). The Motion-Lab is a part of the infrastructure in the Norwegian Center for Offshore Wind Energy (NORCOWE) [7] and is together with UiA and NORCOWE, founded by the Research Council of Norway. The main aim of the Motion-Lab is focused towards extending and improving the current methods for offshore motion compensation. Examples of experiments which can be conducted with the facility are; load transfer operations (vessel-to-fixed and vessel-to-vessel), accuracy performance evaluation of measurement systems (LIDARS, MRU's, etc.), playback of 6 degree-of-freedom (DOF) time series (wave-induced vessel motion), etc. The Motion-Lab is open for researchers, students and external partners to conduct related experiments, simulations, and measurements.

The main equipment installed in the Motion-Lab are the two Stewart platforms, intended for simulation of two floating vessels exposed to stochastic wave motion. An industrial 6-DOF robot is mounted on top of the larger platform, which enables experiments with load handling operation. This type of configuration enables experiments related to vessel-to-vessel motion compensation (VVMC) to be conducted in the laboratory. The facility is equipped with multiple sensors and tracking equipment, allowing for extensive motion measurement. The Motion-Lab is a unique testing facility, and several successful experiments have already been carried out in the laboratory [3, 4, 8].

2.1 Equipment and Setup

A photo of the Motion-Lab is shown in Fig. 2.1. The two Stewart Platforms from Bosch Rexroth (E-Motion 1500 and E-Motion 8000) are visible, and the industrial robot (COMAU Smart 5 NJ110 - 3.0) can be seen mounted on top of the larger platform. Motion reference units (MRUs) from Kongsberg Seatex are installed on each platform and a motion capture system consisting of 17 Oqus 700+ cameras from Qualisys can be seen mounted to the surrounding walls. A laser tracker (LEICA Absolute Tracker AT960) is also available for use. All of the mentioned equipment, except for the motion capture system, are connected either directly or indirectly to an embedded PC (Beckhoff CX 2040) which acts as the central control unit for the Motion-Lab.

Figure 2.1: An Overview of the Norwegian Motion Laboratory [6]

2.1.1 Stewart Platforms

As mentioned, the Motion-Lab is equipped with two 6-DOF E-Motion Platforms, these platforms are known as Stewart Platforms which is a type of parallel robot. Each platform has six prismatic electrical actuators. These actuators are installed in pairs at three fixed corners of the base and are linked with a neighboring actuator at the platform. This configuration enables the platform to move in 6 degrees-of-freedom. Popular fields of application for the Stewart Platform includes flight simulations, wave motion simulations, and driving simulators.

E-Motion 8000

The largest Stewart Platform is a Bosch Rexroth E-Motion 8000 (EM 8000), this motion system has a payload capacity of 8000 $[kg]$. For the ongoing experiments related to VVMC, the EM 8000 is most often used to simulate the motion of the main vessel. The platform is connected to a motion computer, later referred to as *Motion PC 1*, which is controllable via the central control unit. Technical specification and capacity of the EM 8000 can be found in App. A.1.1 (Tab. A.1). A photo of the EM 8000 is shown in the figure (Fig. 2.2)

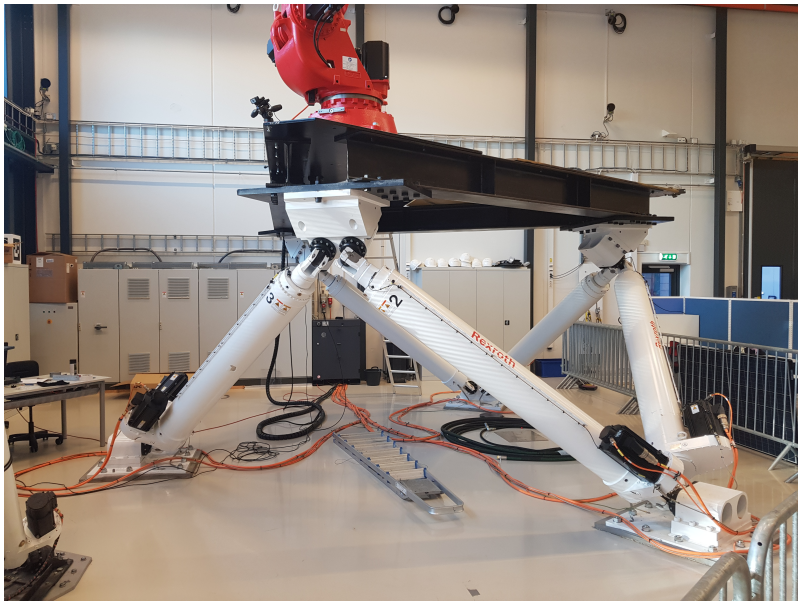


Figure 2.2: Photo of Stewart Platform EM 8000

E-Motion 1500

A Bosch Rexroth E-Motion 1500 (EM 1500) is the smaller Stewart Platform, it has a load capacity of 1500 $[kg]$ and is often used in experiments to simulate the secondary vessel in VVMC or the receiving platform in load handling operations. In the same manner as the larger platform, the EM 1500 is connected to an independent motion computer (*Motion PC 2*) which can be controlled by the central control unit. A photo of the EM 1500 platform is shown in the figure below (Fig. 2.3), and a table of the technical specification can be found in App. A.1.2 (Tab. A.2).

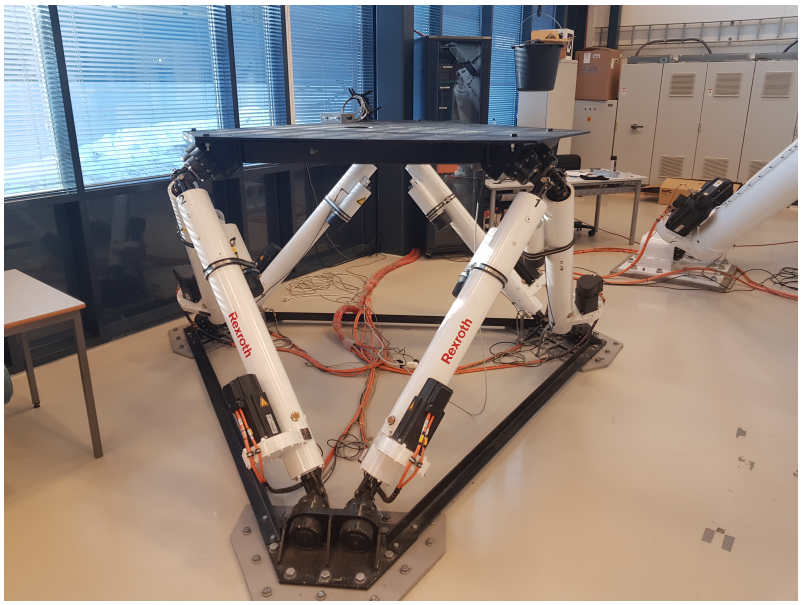


Figure 2.3: Photo of Stewart Platform EM 8000

2.1.2 Industrial Robot

The industrial robot associated with the Motion-Lab is a Comau Smart 5 NJ 110-3.0, this high-performance industrial robot is installed on top of the EM 8000, which is shown in Fig. 2.4. The Comau robot is a six-DOF elbow manipulator which utilizes a parallelogram linkage design. Meaning that the actuator responsible for the elbow motion is localized on the shoulder, which reduces the weight of the upper arm. It has a wrist payload capacity of 110 $[kg]$, and a maximum horizontal reach close to 3 $[m]$ [9]. The robot is connected to an independent computer running a Linux operating system, which is controllable from the central control unit. Related to the experiments conducted in the Motion-Lab, the Comau robot is often used to simulate load handling and crane operations. To undergo these scenarios it is possible to lock the wrist joints of the Comau robot, hence removing 3-DOF from the robot, this enables simulations with a closer resemblance with an offshore crane. Additional technical specification and information of the work-space of the Comau robot can be found in App. A.1.2.

Figure 2.4: Photo of the Comau Smart 5 NJ 110-3.0

2.1.3 Electrical Winch

To simulate the lifting operation of load handling scenarios, an electrically actuated winch is installed on top of the industrial robot, see Fig. 2.4. The winch drum is actuated by a Servo Motor from Beckhoff (AM8532-H) [10], which is controlled by a digital servo drive from Beckhoff (AX5103) [11]. The drive is connected via EtherCAT to the central control unit, which makes it fully controllable through the Beckhoff interface. Some technical information of the servomotor (winch motor) is listed in Tab. 2.1.

Table 2.1: Technical Information of the Beckhoff Servomotor (AM8532-H). Data for 400 V AC [10]

Description	**Value**	**Unit**
Standstill Torque	2.37	$[Nm]$
Rated Torque	1.85	$[Nm]$
Peak Torque	11.65	$[Nm]$
Rated Power	1.38	$[kW]$
Rated Speed	9000	$[rpm]$

2.1.4 Motion Reference Unit

The two Stewart Platforms are equipped with a motion reference unit (MRU H 5th generation) from Kongsberg Seatex [12]. These MRU's incorporates three highly accurate accelerators and three high-end angular rate gyros, which enables 6-DOF measurement for the motion of each platform. Both of the MRU's are directly connected to the central control unit, enabling for easy access to the motion data. A picture of the Kongsberg Seatex MRU installed on the EM 1500 platform can be seen in 2.5.

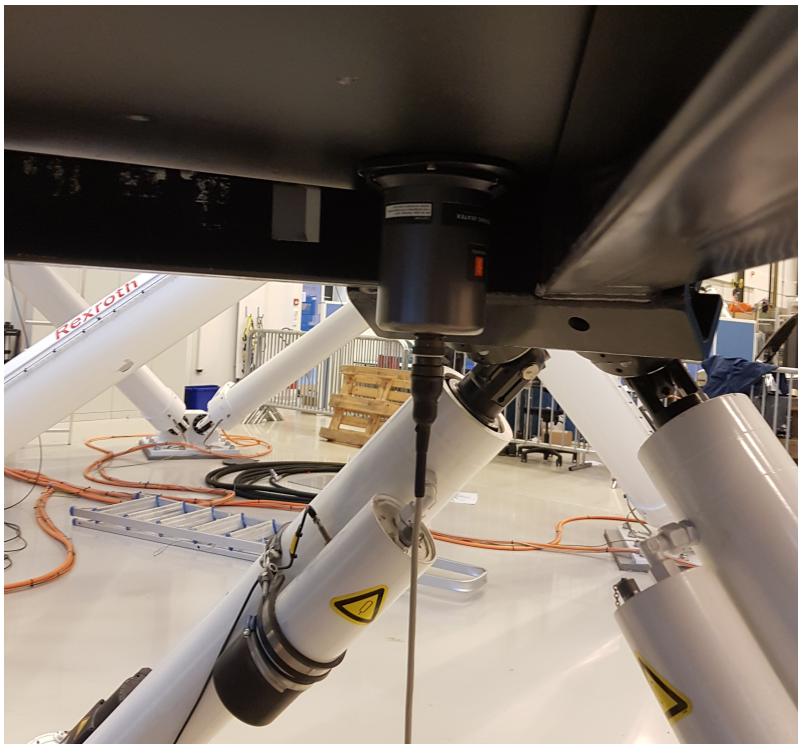


Figure 2.5: Photo of the Kongsberg Seatex MRU Installed on EM 1500

2.1.5 Motion-Capture System

A full motion-capture system is available in the Motion-Lab. 17 Qqus 700+ cameras from Qualisys [13] are mounted on the surrounding walls, which enables a full coverage of the laboratory's working volume. With the use of markers, the cameras can record and calculate position with high accuracy and speed. In *normal mode* the cameras can capture 300 frames per second (FPS) with 12 megapixels (MP) and 4096 x 3072 resolution. Enabling *High-speed mode* the frame rate enhances to 1100 FPS with 3 MP and 2024 x 1536 resolution. The data captured by the cameras are logged with a Qualisys motion-capture system which runs on a dedicated computer. Fig. 2.6 shows pictures of the wall mounted Qqus cameras.

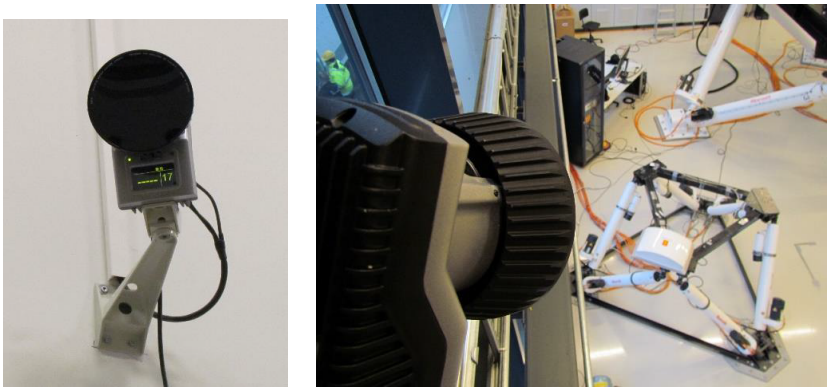


Figure 2.6: Motion-Capture System - Oqus 700+ Cameras

2.1.6 Laser Tracker

To perform high-speed dynamic measurements, the Motion-Lab is equipped with a Leica Absolute Tracker AT960. The laser tracker can deliver high-precision 6-DOF probing and measurement data, it offers a maximum permissible error of $\pm 15\mu m + 6\mu m \backslash m$ and a measuring range up to $160m$ (ø) [14]. In addition to the laser tracker, a Leica T-Mac Frame (TMC30-F) 6-DOF measuring probe is also available in the Motion-Lab. Combining this with the Leica AT960 enables real-time tracking capabilities in 6-DOF. The laser tracker can in this case follow the probe, and give high-speed automated measurements with a sampling frequency of $1kHz$. A picture of the Leica AT960 laser tracker and the Leica T-Mac probe can be seen in Fig. 2.7

(a) Leica Absolute Tracker (AT960) (b) Leica T-Mac Frame (TMC30-F)

Figure 2.7: Photos of the Laser Tracker and Measuring Probe [14]

2.1.7 Central Control Unit

Acting as a central control unit for the equipment installed in the Motion-Lab is a Beckhoff CX 2040. The control unit is a combination of an embedded computer and a hardware PLC which runs on a Windows 7 Embedded based operating system (OS). With Beckhoff TwinCAT automation software [15], it is possible to configure the control unit to run a real-time runtime kernel in parallel with the Windows OS, known as TwinCAT - eXtended Automation Runtime (XAR) [16]. This feature removes normal OS interrupts and guarantees real-time control and monitoring of the connected equipment.

The TwinCAT software is also available with the TwinCAT - eXtended Automation Engineering (XAE) which works as an extension to Microsoft Visual Studio. This feature enables the user to develop programs on a standard Windows computer (with Visual Studio) using IEC6113-3 and C or C++ languages and to upload the program to a connected Beckhoff unit. An illustration of the XAE scheme is shown in Fig. 2.8. The *TwinCAT Standard* enables the use of the basic framework of Visual Studio with the benefits concerning handling, connection to source code control software, etc. The *Twincat Integrated* will integrate itself into Visual Studio, which makes C, C++, C# programming languages and links to Matlab/Simulink available [17].

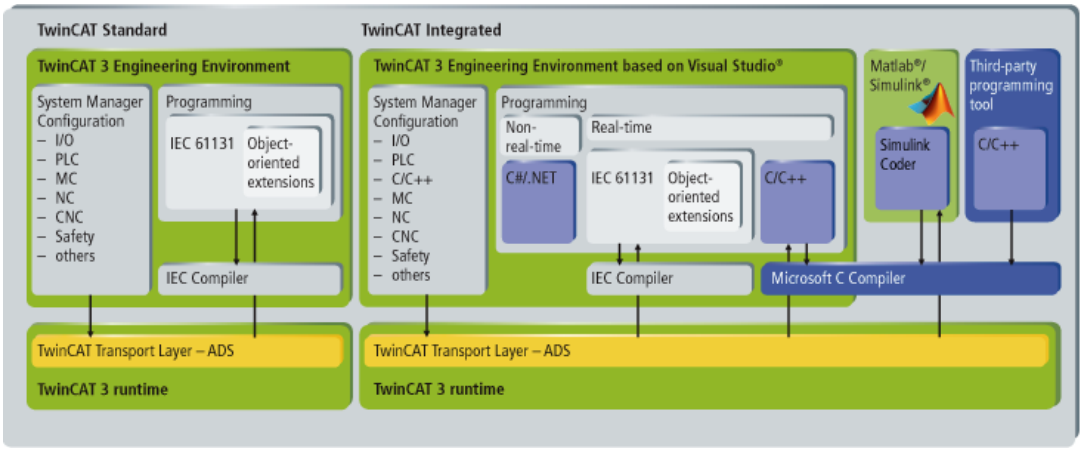


Figure 2.8: TwinCAT 3 XAE Intergration [17]

The control unit communicates with the connected equipment through either an Ethernet or EtherCAT connection, or directly through the I/O modules. During real-time runtime, data can be transmitted and received, to and from the control unit via ADS router. In other words, while the control unit runs the assigned program on the real-time kernel, the development computer (standard Windows computer, non-real-time) can communicate with the control unit by sending and receiving data via the ADS protocol. An illustration of the ADS communication bridge is shown in Fig. 2.9.

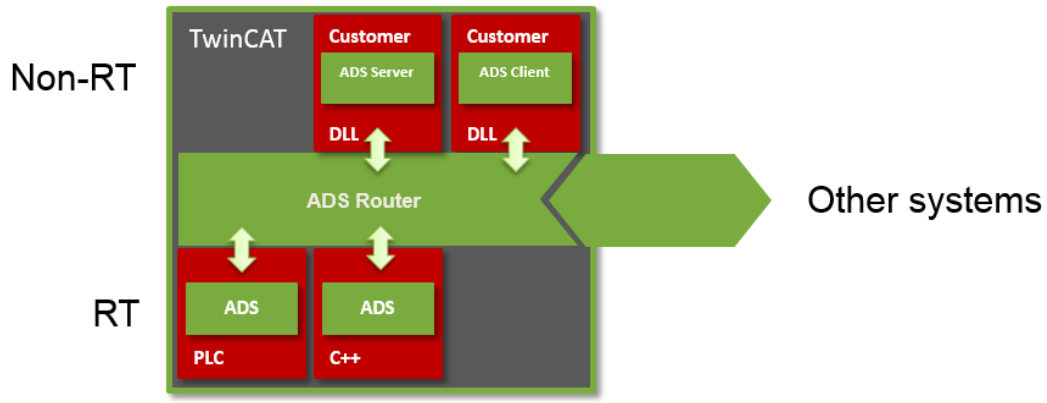


Figure 2.9: TwinCAT ADS Communication Brigde [18]

Similar to a standard PLC, the Beckhoff CX 2040 has available slots for installation of extension modules. The control unit available in the Motion-Lab is equipped with a total of 6 analog and digital I/O modules; two analog output modules, two analog input modules, one digital input module and one digital output module. Fig. 2.10 presents an overview of the installed modules and the related connection configuration.

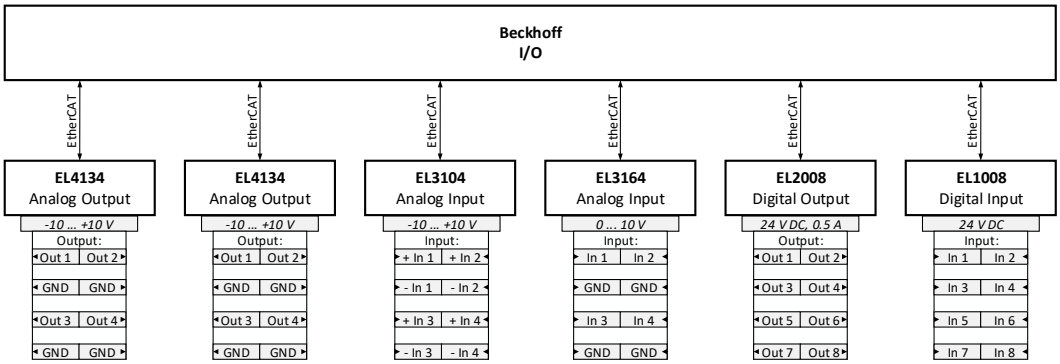


Figure 2.10: Configuration of the Analog and Digital I/O Modules of the Beckhoff PC/PLC

2.2 Communication

As presented in Sec. 2.1, the Norwegian Motion Laboratory consists of multiple different equipment and systems. This also includes different computers which are responsible for the connection and interaction with the equipment, where the Beckhoff embedded PC/PLC is working as the overall central control unit. This section will present an overview of the equipment connected to the control unit, and the communication configuration.

Currently, a total of 6 computers are installed in the Motion-Lab:

- The central control unit, a Beckhoff embedded PC/PLC.
- Two motion PC's, one for each Stewart Platform (EM 1500 and EM 8000).
- A Linux based system, for the interaction with the Comau industrial robot.
- A dedicated motion-capture computer, for the Qualisys Qqus motion-capture system.
- A development computer running a standard Windows OS, which is referred to as the Host PC.

A schematic of the current communication configuration is shown in Fig. 2.11. The lowest level illustrates the connected equipment to the central control unit, where some are indirectly connected via dedicated computers. It should be noted that the Qualisys motion-capture system is currently not connected to the control unit and therefore not included in the schematics.

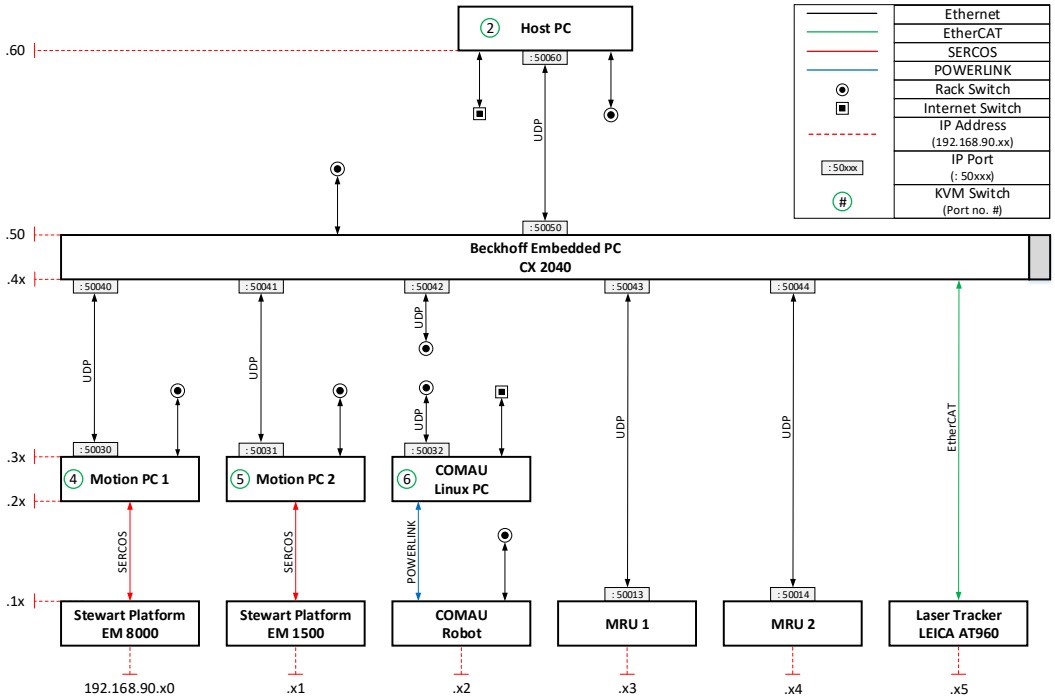


Figure 2.11: Overview of the Communication Configuration of the Motion-Lab

As the communication overview illustrates, different connections use different communication protocols, e.g., the Stewart Platforms use a SERCOS protocol to communicate with the motion PC's, and User Datagram Protocol (UDP) is used for communication between the MRU and the control unit. The communication protocols are illustrated by different colors of the vertical connection lines, where some require a specific communication cable and connection socket.

Each of the installed components has been assigned one or more static IP address and a communication port. All of the connected equipment is assigned an IP address on the form: [192.168.xx]. Here the two last digits [xx] represents the row and column number, respectively. This is illustrated with the dashed red lines. An example of this is the *Comau Robot*, which is located at the first row, and the third column, hence it has been allocated with [192.168.13] as its IP address. It should be noted that some of the installed equipment has several communication interfaces, which leads to multiple IP addresses being assigned to the same system. E.g. *Motion PC 2* has one interface for communication with the EM 1500 [192.168.21], and one for UDP communication with the central control unit [192.168.31].

There are two network switches installed in the communication configuration. The *Internet Switch* is for internet connection, where the connected systems are assigned a dynamic IP address. A *Rack Switch* is also installed in the lab, which is used for internal connection between the equipment. This enables for easier interaction between the equipment and the development computer. The equipment connected to the rack switch has been assigned with an additional static IP address on the form: [192.168.1xx], where the two last digits correlate to the row and column number. As an example, *Motion PC 1* has three different communication interfaces. [192.168.20] is assigned for the connection with the Stewart Platform, [192.168.30] for the UDP connection with the control unit (Beckhoff CX 2040), and [192.168.130] is assigned for the connection to the *Rack Switch*.

All of the previously presented computers are connected to a keyboard-video-mouse (KVM) switch, except the Beckhoff embedded PC/PLC (this is controlled by a remote desktop configuration). The KVM switch allows the user to interact with the available computers by a master keyboard, mouse, and screen. The numbers outlined by green circles (see Fig. 2.11) relates to the assigned KVM switch port number.

3 Theory

This chapter will present the fundamental theory and governing equations applied in the future approaches and methods. Which include topics such as robot kinematics, load dynamics, control system design and implementation.

3.1 Robot Kinematics

Kinematics is known as the description of a manipulators motion, without considering the required torques and forces. This section will describe the use of forward and inverse kinematics, to derive a set of governing equations which describes the relation between the joint variables and the motion and orientation of the industrial robot.

3.1.1 Robot Geometry and Dimensions

In this project, the industrial robot is used to simulate a 3-DOF offshore loader crane. Meaning that the wrist joints of the robot (3-DOF) will be considered as fixed, and will not be actuated. Fig. 3.1 shows an overview of the industrial robot with the installed extension tool, together with the notation of main dimensions and coordinate systems of the joints.

As Fig. 3.1 shows, the Comau robot utilizes a parallelogram linkage design. This design allows the actuator for the elbow motion to be positioned at the shoulder of the robot, which reduces the weight of the upper arm. A general principle in robot design is to locate as much of the robot mass away from the distal links [19]. In this thesis, the dynamics of the robot are not taken into consideration. The closed-chain of the parallelogram link is therefore simplified, and the elbow actuator is considered to be located at the elbow joint (x_3, y_3, z_3). This simplification allows the robot to be treated as an open-chain manipulator, which can be considered to be valid, due to the parallelogram linkage is kinematically equivalent to a two-link planar arm [20]. The dimension values of the Comau robot are listed in Tab. 3.1.

Table 3.1: Comau Robot Dimensions

Notation	Value	Unit
a_1	0.350	$[m]$
d_1	0.830	$[m]$
a_2	1.160	$[m]$
a_3	0.250	$[m]$
d_4	1.492	$[m]$
d_6	0.210	$[m]$
d_{tp}	0.567	$[m]$

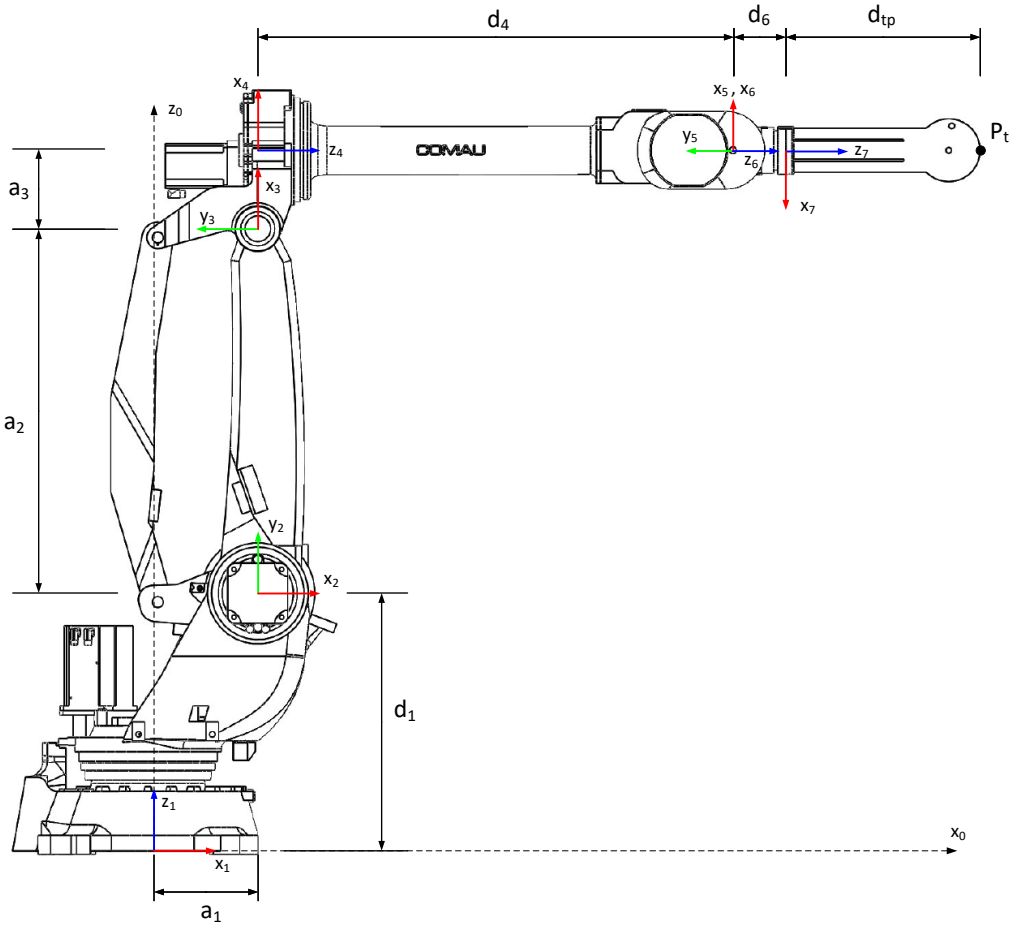


Figure 3.1: Geometry, Dimensions and Joint Coordinate System of the Comau Robot

3.1.2 Forward Kinematics

Forward kinematics is used to determine the pose of the manipulator end effector as a function of the joint angles. For the industrial robot, it is desired to find a set of equations which can describe the position, velocity, and acceleration of the tool-point using the angular position, velocity and acceleration of the joints as inputs.

A convention known as *Denavit-Hartenberg* (DH) is used to derive the forward kinematics for an open-chain manipulator [19]. This technique is a systematic and commonly used approach for describing the pose of the tool-point based on the joint angles. The DH parameter table is constructed by assigning a local right-handed coordinate system in each joint, with the z-axis aligned with the rotational axis of the joint. Then a parameter table can be constructed by representing each homogeneous transformation A_i as a product of four basic transformations:

$$A_i = R_z(\theta_i)T_z(d_i)T_x(a_i)R_x(\alpha_i) \tag{3.1}$$

Where:

$$R_z(\theta_i) = \begin{bmatrix} \cos\theta_i & -\sin\theta_i & & 0 \\ \sin\theta_i & \cos\theta_i & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad T_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$T_x(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \qquad R_x(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\alpha_i & -\sin\alpha_i & 0 \\ 0 & \sin\alpha_i & \cos\alpha_i & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The parameters θ_i, d_i, a_i, α_i are associated with link i and joint i, and are referred to as; joint angle, link offset, link twist and link length, respectively. These parameters are given by the relationship between the two chosen coordinate systems, where for a revolute joint the θ_i parameter is the single variable, and the remaining three are constants.

In this thesis, the right-hand rule convention is used for coordinate system and joint orientation. The interface provided by Comau uses clockwise rotation as positive, with the forearm pointing upwards (link d_4 parallel to z_0, see Fig. 3.1) as default home position. Correcting for these disparities, a DH parameter table of the Comau robot has been constructed using the overview shown in Fig. 3.1 and is presented in Tab. 3.2. Notice that the coordinate system of joint 1 is positioned at the location of the global coordinate system (x_0,y_0,z_0), hence no relative translation or rotation, between them.

Table 3.2: Denavit-Hartenberg Table for Comau Robot (3-DOF)

Link i	θ_i	d_i	a_i	α_i
1	$-\theta_1$	d_1	a_1	$\frac{\pi}{2}$
2	$\frac{\pi}{2} - \theta_2$	0	a_2	0
3	$\theta_3 + \frac{\pi}{2} + \theta_2$	0	a_3	$\frac{\pi}{2}$
4	π	$d_4 + d_6 + d_{tp}$	0	0

As mentioned in Sec. 3.1.1, the Comau robot is in this project considered as a 3-DOF system with the wrist joints considered fixed. Hence, the presented DH table (Tab. 3.2) does not include all available joints. A full DH parameter table for the Comau robot, with correction for the disparities of the Comau interface, is available in App. A.2.

Using the constructed DH table (Tab. 3.2) the homogeneous transformation matrices can be derived as follows:

$$\begin{aligned} A_1 &= R_z(-\theta_1)T_z(d_1)T_x(a_1)R_x(\frac{\pi}{2}) \\ &= \begin{bmatrix} \cos\theta_1 & 0 & -\sin\theta_1 & a_1\cos\theta_1 \\ -\sin\theta_1 & 0 & -\cos\theta_1 & -a_1\sin\theta_1 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix} \end{aligned} \tag{3.2}$$

$$A_2 = R_z(\frac{\pi}{2} - \theta_2)T_z(0)T_x(a_2)R_x(0)$$
$$= \begin{bmatrix} \sin\theta_2 & -\cos\theta_2 & 0 & a_2\sin\theta_2 \\ \cos\theta_2 & \sin\theta_2 & 0 & a_2\cos\theta_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$A_3 = R_z(\theta_3 + \frac{\pi}{2} + \theta_2)T_z(0)T_x(a_3)R_x(\frac{\pi}{2})$$
$$= \begin{bmatrix} -\sin(\theta_2+\theta_3) & 0 & \cos(\theta_2+\theta_3) & -a_3\sin(\theta_2+\theta_3) \\ \cos(\theta_2+\theta_3) & 0 & \sin(\theta_2+\theta_3) & a_3\cos(\theta_2+\theta_3) \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$A_4 = R_z(\pi)T_z(d_4 + d_6 + d_{tp})T_x(0)R_x(0)$$
$$= \begin{bmatrix} -1 & 0 & 0 & 0 \\ 0 & -1 & 0 & 0 \\ 0 & 0 & 1 & d_4 + d_6 + d_{tp} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

Now by using Eq. 3.2 - Eq. 3.5 and substitute the forearm length with $L = d_4 + d_6 + d_{tp}$, the transformation matrix of the robot tool-point relative to the global coordinate can be derived as shown by Eq. 3.6.

$$T_t^0 = T_4^0 = A_1A_2A_3A_4 \tag{3.6}$$
$$= \begin{bmatrix} \cos\theta_1\cos\theta_3 & \sin\theta_1 & -\cos\theta_1\sin\theta_3 & \cos\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ -\cos\theta_3\sin\theta_1 & \cos\theta_1 & \sin\theta_1\sin\theta_3 & -\sin\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ \sin\theta_3 & 0 & \cos\theta_3 & d_1 + L\cos\theta_3 + a_2\cos\theta_2 - a_3\sin\theta_3 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The first 3×3 entries of Eq. 3.6 contains information of the orientation of the robot tool-point relative to the global coordinate, this is also referred to as the rotation matrix R_t^0.

$$R_t^0 = \begin{bmatrix} \cos\theta_1\cos\theta_3 & \sin\theta_1 & -\cos\theta_1\sin\theta_3 \\ -\cos\theta_3\sin\theta_1 & \cos\theta_1 & \sin\theta_1\sin\theta_3 \\ \sin\theta_3 & 0 & \cos\theta_3 \end{bmatrix} \tag{3.7}$$

Tool-Point Position

The first three entries of the last column of Eq. 3.6 describes the position of the tool-point relative to the global coordinate system. Which means that the forward kinematics equation for the tool-point position is equal to:

$$P_t = \begin{bmatrix} x_t \\ y_t \\ z_t \end{bmatrix} = \begin{bmatrix} \cos\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ -\sin\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ d_1 + L\cos\theta_3 + a_2\cos\theta_2 - a_3\sin\theta_3 \end{bmatrix} \tag{3.8}$$

Where:

x_t	-	Tool-point position, x-direction	$[m]$
y_t	-	Tool-point position, y-direction	$[m]$
z_t	-	Tool-point position, z-direction	$[m]$
θ_1	-	Angular position, joint 1	$[rad]$
θ_2	-	Angular position, joint 2	$[rad]$
θ_3	-	Angular position, joint 3	$[rad]$
a_1	-	Dimension	$[m]$
a_2	-	Dimension	$[m]$
d_1	-	Dimension	$[m]$
L	-	Dimension ($L = d_4 + d_6 + d_{tp}$)	$[m]$

Jacobian Matrix

The Jacobian matrix is an important quantity in the analysis of robot kinematics and works as a mapping between the joints angular velocity and tool-point velocity. The Jacobian matrix is derived from the forward kinematics equation Eq. 3.8, and is expressed as:

$$J = \begin{bmatrix} \frac{\partial x_t}{\partial \theta_1} & \frac{\partial x_t}{\partial \theta_2} & \frac{\partial x_t}{\partial \theta_3} \\ \frac{\partial y_t}{\partial \theta_1} & \frac{\partial y_t}{\partial \theta_2} & \frac{\partial y_t}{\partial \theta_3} \\ \frac{\partial z_t}{\partial \theta_1} & \frac{\partial z_t}{\partial \theta_2} & \frac{\partial z_t}{\partial \theta_3} \end{bmatrix} \tag{3.9}$$

Inserting the full expression of Eq. 3.8 into Eq. 3.9 gives the following Jacobian matrix.

$$J = \begin{bmatrix} J_{11} & J_{12} & J_{13} \\ J_{21} & J_{22} & J_{23} \\ J_{31} & J_{32} & J_{33} \end{bmatrix} \tag{3.10}$$

Where the entries of Eq. 3.10 are equal to

$$\begin{aligned} J_{11} &= -\sin\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ J_{12} &= a_2\cos\theta_1\cos\theta_2 \\ J_{13} &= -\cos\theta_1(L\cos\theta_3 - a_3\sin\theta_3 \\ J_{21} &= -\cos\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\ J_{22} &= -a_2\cos\theta_2 sin\theta_1 \\ J_{23} &= \sin\theta_1(L\cos\theta_3 - a_3\sin\theta_3) \\ J_{31} &= 0 \\ J_{32} &= -a_2\sin\theta_2 \\ J_{33} &= -L\sin\theta_3 - a_3\cos\theta_3 \end{aligned}$$

Tool-Point Velocity

With the derived Jacobian matrix (Eg. 3.10), the forward kinematic equation for the tool-point velocity as a function of the joint's angular velocity can be expressed.

$$\dot{P}_t = \begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix} = J \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{3.11}$$

Where:

$\dot{x}_t$	-	Tool-point velocity, x-direction	$[\frac{m}{s}]$
$\dot{y}_t$	-	Tool-point velocity, y-direction	$[\frac{m}{s}]$
$\dot{z}_t$	-	Tool-point velocity, z-direction	$[\frac{m}{s}]$
J	-	Jacobian matrix	$[-]$
$\dot{\theta}_1$	-	Angular velocity, joint 1	$[\frac{rad}{s}]$
$\dot{\theta}_2$	-	Angular velocity, joint 2	$[\frac{rad}{s}]$
$\dot{\theta}_3$	-	Angular velocity, joint 3	$[\frac{rad}{s}]$

Derivative of the Jacobian Matrix

In the same manner as the Jacobian matrix relates to the mapping of the angular joints and tool-point velocity, the derivative of the Jacobian matrix aids to derive the relation between joint angular acceleration and tool-point acceleration. The derivative of the Jacobian matrix can be derived by differentiating the original Jacobian matrix (Eg. 3.10), or by the following expression:

$$\dot{J} = \begin{bmatrix} \frac{\partial(J\dot{\theta}_1)}{\partial\theta_1} & \frac{\partial(J\dot{\theta}_1)}{\partial\theta_2} & \frac{\partial(J\dot{\theta}_1)}{\partial\theta_3} \\ \frac{\partial(J\dot{\theta}_2)}{\partial\theta_1} & \frac{\partial(J\dot{\theta}_2)}{\partial\theta_2} & \frac{\partial(J\dot{\theta}_2)}{\partial\theta_3} \\ \frac{\partial(J\dot{\theta}_3)}{\partial\theta_1} & \frac{\partial(J\dot{\theta}_3)}{\partial\theta_2} & \frac{\partial(J\dot{\theta}_3)}{\partial\theta_3} \end{bmatrix} \tag{3.12}$$

The derivative of the Jacobian matrix for the Comau robot can be expressed as.

$$\dot{J} = \begin{bmatrix} \dot{J}_{11} & \dot{J}_{12} & \dot{J}_{13} \\ \dot{J}_{21} & \dot{J}_{22} & \dot{J}_{23} \\ \dot{J}_{31} & \dot{J}_{32} & \dot{J}_{33} \end{bmatrix} \tag{3.13}$$

Where each entry of Eq. 3.13 are equal to

$$
\begin{aligned}
\dot{J}_{11} &= -\dot{\theta}_1 \cos\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\
&\quad - a_2\dot{\theta}_2 \sin\theta_1 \cos\theta_2 \\
&\quad + \dot{\theta}_3 \sin\theta_1(L\cos\theta_3 - a_3\sin\theta_3) \\
\dot{J}_{12} &= -a_2\dot{\theta}_1 \sin\theta_1 \cos\theta_2 \\
&\quad - a_2\dot{\theta}_2 \cos\theta_1 \sin\theta_2 \\
\dot{J}_{13} &= \dot{\theta}_1 \sin\theta_1(L\cos\theta_3 - a_3\sin\theta_3) \\
&\quad + \dot{\theta}_3 \cos\theta_1(L\sin\theta_3 + a_3\cos\theta_3) \\
\dot{J}_{21} &= \dot{\theta}_1 \sin\theta_1(a_1 - L\sin\theta_3 - a_3\cos\theta_3 + a_2\sin\theta_2) \\
&\quad - a_2\dot{\theta}_2 \cos\theta_1 \cos\theta_2 \\
&\quad + \dot{\theta}_3 \cos\theta_1(L\cos\theta_3 - a_3\sin\theta_3) \\
\dot{J}_{22} &= -a_2\dot{\theta}_1 \cos\theta_1 \cos\theta_2 \\
&\quad + a_2\dot{\theta}_2 \sin\theta_1 \sin\theta_2 \\
\dot{J}_{23} &= \dot{\theta}_1 \cos\theta_1(L\cos\theta_3 - a_3\sin\theta_3) \\
&\quad - \dot{\theta}_3 \sin\theta_1(L\sin\theta_3 + a_3\cos\theta_3) \\
\dot{J}_{31} &= 0 \\
\dot{J}_{32} &= -a_2\dot{\theta}_2 \cos\theta_2 \\
\dot{J}_{33} &= -\dot{\theta}_3(L\cos\theta_3 - a_3\sin\theta_3)
\end{aligned}
$$

Tool-Point Acceleration

With the derivative of the Jacobian matrix (Eq. 3.13), it is possible to derive the forward kinematic expression for the tool-point acceleration, as a function of the joint's angular velocity and acceleration.

$$
\ddot{P}_t = \begin{bmatrix} \ddot{x}_t \\ \ddot{y}_t \\ \ddot{z}_t \end{bmatrix} = \dot{J} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} + J \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} \tag{3.14}
$$

Where:

$\ddot{x}_t$	-	Tool-point acceleration, x-direction	$[\frac{m}{s^2}]$
$\ddot{y}_t$	-	Tool-point acceleration, y-direction	$[\frac{m}{s^2}]$
$\ddot{z}_t$	-	Tool-point acceleration, z-direction	$[\frac{m}{s^2}]$
$\dot{J}$	-	Derivative of the Jacobian matrix	$[-]$
$\ddot{\theta}_1$	-	Angular acceleration, joint 1	$[\frac{rad}{s^2}]$
$\ddot{\theta}_2$	-	Angular acceleration, joint 2	$[\frac{rad}{s^2}]$
$\ddot{\theta}_3$	-	Angular acceleration, joint 3	$[\frac{rad}{s^2}]$

The Matlab scripts used to derive the presented governing equations for the forward kinematics can be found in App. C.1.1.

3.1.3 Inverse Kinematics

Inverse kinematic concerns the derivation of the joint angular configuration given a desired tool-point position. In other words, a set of governing equations describing the required robot joint variables needed to obtain a desired tool-point motion. In this thesis, a geometric approach will be used to derive the inverse kinematics, and since the robot will be considered as a 3-DOF system, the inverse kinematics of the joints related to the wrist motion will be omitted.

Joint Angle θ_1

The 3-DOF robot can be considered as the configuration shown in Fig. 3.2. The tool-point coordinates have been projected onto the XY-plane of the global coordinate system (x_0, y_0, z_0).

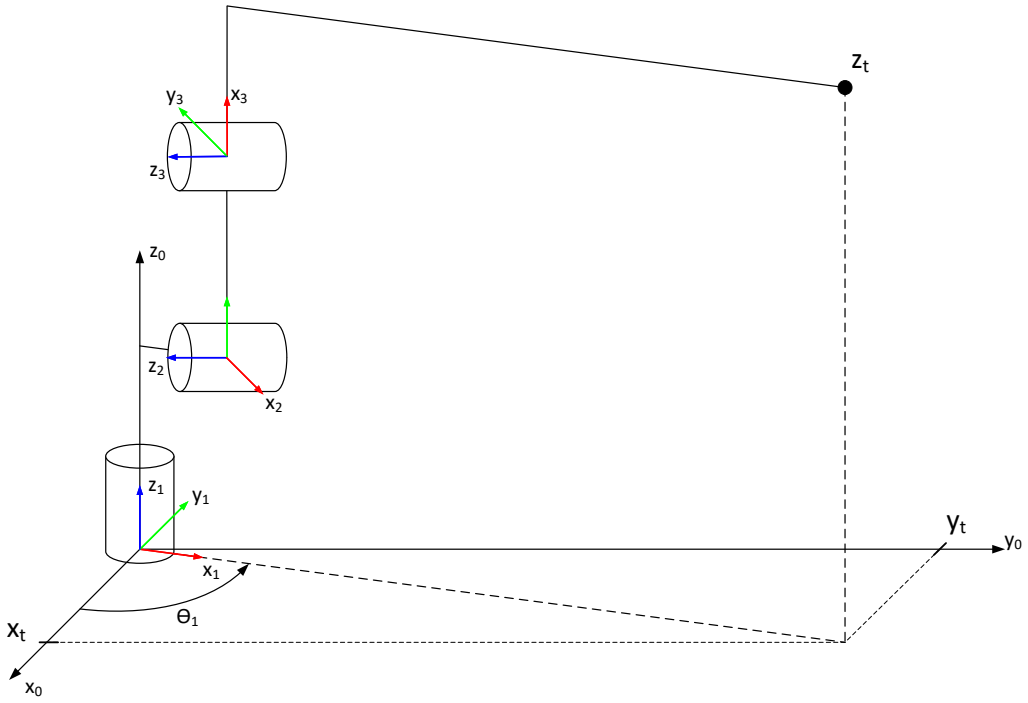


Figure 3.2: Robot Base Rotation based on Projection of the Tool-Point

Using this projection, the robot base can be seen to have rotated an angle θ_1, relative to the global coordinate system. This gives an expression of the rotation angle as a function of the tool-point position.

$$\theta_1 = -atan2(y_t, x_t) \tag{3.15}$$

Instead of the normal inverse tangent function, **atan2** will be used in these formulations. The latter function considers the sign of the vector components and place the calculated angle in the correct quadrant.

Joint Angle θ_2

With the base rotation covered, the robot can be simplified and considered as a planar two-link elbow system. Inverse kinematics of such a system has two possible solutions; elbow-down and elbow-up, as can be seen in Fig. 3.3.

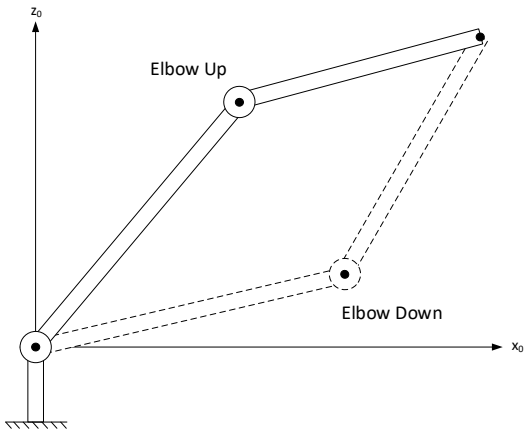


Figure 3.3: Planar Two-Link Elbow Configuration

Due to the structure of the Comau robot, the planar projection differs from the one shown in Fig. 3.3. With the forearm having an offset from the elbow joint. An illustration of the Comau configuration is shown in Fig. 3.4. Here the link a_1 together with the line B symbolises the two-link elbow system used for elbow-down vs. elbow-down analysis.

The length of the line B is constant and can be calculated as:

$$B = \sqrt{{a_3}^2 + L^2} \tag{3.16}$$

The tool-point components ${x_t}'$ and ${y_t}'$ indicates the tool-point position relative to the coordinate system of the shoulder joint (x_2, y_2, z_2). These components can be found by using the transformation matrix.

$$P_t^2 = \begin{bmatrix} {x_t}' \\ {y_t}' \\ {z_t}' \end{bmatrix} = (A_1)^{-1} P_t \tag{3.17}$$

Now the length of line C can be calculated to be:

$$C = \sqrt{({x_t}')^2 + ({y_t}')^2} \tag{3.18}$$

Introducing the law of cosines on the triangle a_2BC gives the following expression.

$$C^2 = {a_2}^2 + B^2 - 2a_2 B\cos(\pi - \alpha) \tag{3.19}$$

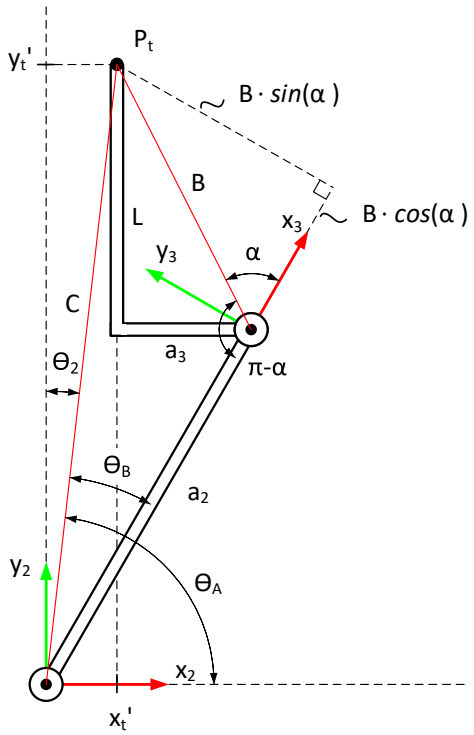


Figure 3.4: Geometric Approach for Inverse Kinematics of θ_2

Using the properties of the unit circle ($\cos(\pi - \alpha) = -\cos\alpha$) Eq. 3.19 can be rewritten as:

$$\cos\alpha = \frac{C^2 - {a_2}^2 - B^2}{2a_2 B} := D \tag{3.20}$$

D is defined as equal to $\cos\alpha$ for easier notation. An expression for α can now be derived from Eq. 3.20, but a better approach is to introduce the Pythagorean identity.

$$\sin^2\alpha + \cos^2\alpha = 1 \tag{3.21}$$

$$\Rightarrow \sin\alpha = \pm\sqrt{1 - D^2} \tag{3.22}$$

Now combining Eq. 3.20 and Eq. 3.22, the angle α is equal:

$$\alpha = atan2(\sin\alpha, D) \tag{3.23}$$

$$= atan2(\pm\sqrt{1 - D^2}, D) \tag{3.24}$$

The advantages of expressing the angle α as in Eq. 3.24, lies with the possibility to select the elbow configuration based on the two solutions of the square root. An elbow-down and elbow-up configuration is selected by choosing a negative and a positive sign, respectively.

The two assisting variables θ_A and θ_B can be derived as

$$\theta_A = atan2({y_t}', {x_t}') \tag{3.25}$$

$$\theta_B = atan2(B\sin\alpha, a_2 + B\cos\alpha) \tag{3.26}$$

Finally, the joint angle θ_2 can be expressed.

$$\theta_2 = \frac{\pi}{2} - (\theta_A - \theta_B) \tag{3.27}$$

Joint Angle θ_3

The last joint angle θ_3 is derived based on the same geometric approach. Fig. 3.5 shows the same planar two-link elbow system, with the elbow joint in a small angular offset.

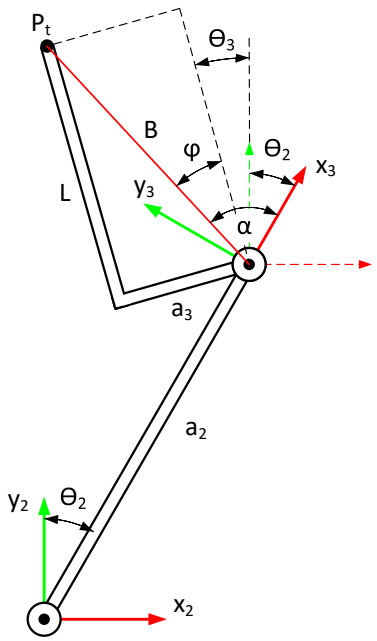


Figure 3.5: Geometric Approach for Inverse Kinematics of θ_3

The constant angle φ can be calculated to be equal:

$$\varphi = atan2(a_3, L) \tag{3.28}$$

Using the geometries shown in Fig. 3.5, and with α and θ_2 expressed by Eq. 3.24 and Eq. 3.27, respectively. The final joint angle θ_3 can be expressed as:

$$\theta_3 = \alpha - \varphi - \theta_2 \tag{3.29}$$

Joint Angular Velocity

In the derivation of the robot forward kinematics, the Jacobian matrix served as an important quantity, which is also true for the inverse kinematics. The joint's angular velocity can be expressed as a product of the inverse Jacobian matrix (found in Eq. 3.10) and the tool-point velocity.

$$\begin{aligned} \dot{\theta} &= J^{-1}\dot{P}_t \\ \Rightarrow \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} &= J^{-1} \begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix} \end{aligned} \tag{3.30}$$

Joint Angular Acceleration

The joint angular acceleration is derived by rearranging Eq. 3.14, which is dependent on the inverse of the Jacobian matrix and the derivative of the Jacobian matrix (Eq. 3.13).

$$\begin{aligned} \ddot{\theta} &= J^{-1}(\ddot{P}_t - \dot{J}\dot{\theta}) \\ \Rightarrow \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} &= J^{-1}\Big(\begin{bmatrix} \ddot{x}_t \\ \ddot{y}_t \\ \ddot{z}_t \end{bmatrix} - \dot{J} \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \Big) \end{aligned} \tag{3.31}$$

3.2 Suspended Load Motion

As a part of this thesis, it is desired to design and develop a system capable of stabilizing the surge and sway motion of a suspended payload. In this regard, it is essential to derive the governing equation for which describes the motion of the load. The suspended load is considered to be connected to the wire of the winch. Hence the payload will be hanging from the robot's tool-point, which will be responsible for the load motion.

This section will present the analysis and derivation of the governing equations for both a 2-dimensional (2D) and a 3-dimensional (3D) system of the suspended load. The former analysis will set a basis for understanding the motion of a hanging load, which will be beneficial when extending the scenario to a full 3D analysis.

The compensation of the suspended load's heave motion is assumed to be achievable by varying the length of the wire, for which the winch motor will act as an actuator. However, this compensation task is not within the scope of this thesis, and will therefore not be analyzed.

3.2.1 Simple Pendulum (2D-System)

Before conducting a full analysis of the 3-dimensional (3D) suspended payload, it was desired to develop a model for the simple pendulum system. The simple hanging pendulum only considers 2-dimensions (2D) but has a significant correlation with the 3D suspended payload. This model will later be used for testing and simulations of different controller schemes and will work as a basis for the upcoming analysis of the full 3D system.

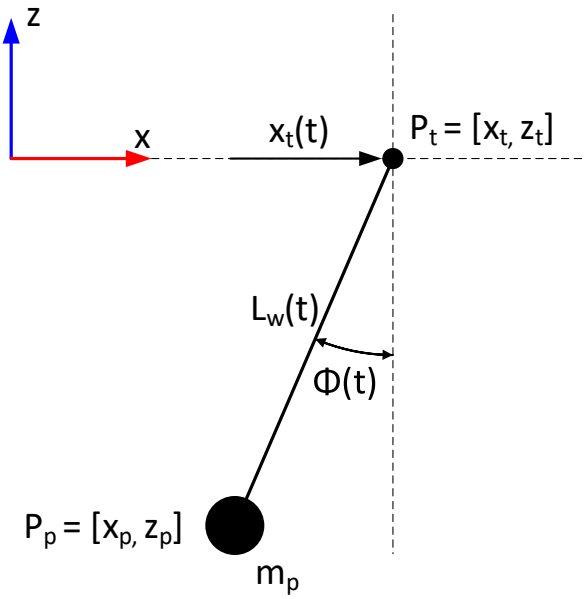


Figure 3.6: An Illustration of the Simple Hanging Pendulum

An illustration of the simple pendulum system is shown in Fig. 3.6. The payload P_p with a mass m_p is connected via a rigid wire with length L_w, to the tool-point P_t. To induce the system, a horizontal motion is applied to the tool-point, which will introduce an angle ϕ between the payload and its equilibrium point.

The analysis of the simple pendulum system is performed under the following assumptions.

- The payload is assumed to be a point mass.
- The wire is assumed to be a massless rigid rod.
- The deflection and elongation of the wire is neglected.
- The frictional elements of the tool-point motion and wire swing is neglected.
- Motion of the tool-point is restricted to the horizontal direction (x).
- The angle ϕ and the tool-point position is assumed to be measurable.

The simple pendulum can be considered as a 3-DOF system, with the generalized coordinate vector $q(t) \in \mathbb{R}^3$.

$$q(t) = \begin{bmatrix} x_t(t) \\ L_w(t) \\ \phi(t) \end{bmatrix} \tag{3.32}$$

Payload Position

The following equation describes the position of the payload coordinates relative to the coordinate system (x,z)

$$x_p = x_t + L_w \sin\phi \tag{3.33}$$
$$z_p = z_t - L_w \cos\phi \tag{3.34}$$

Using Eq. 3.33 and Eq. 3.34, the expression for the payload position can be rewritten in vector formulation.

$$P_p = P_t + L_w \begin{bmatrix} \sin\phi \\ -\cos\phi \end{bmatrix} \triangleq P_t + L_w\lambda \tag{3.35}$$

Payload Velocity

The velocity of the hanging payload is derived by differentiating the expression of the payload position (Eq. 3.35).

$$\dot{P}_p = \dot{P}_t + L_w\dot{\lambda} + \dot{L}_w\lambda \tag{3.36}$$
$$\Rightarrow \begin{bmatrix} \dot{x}_p \\ \dot{z}_p \end{bmatrix} = \begin{bmatrix} \dot{x}_t \\ \dot{z}_t \end{bmatrix} + L_w \begin{bmatrix} \cos\phi \\ \sin\phi \end{bmatrix} \dot{\phi} + \dot{L}_w \begin{bmatrix} \sin\phi \\ -\cos\phi \end{bmatrix}$$

Lagrangian

The Lagrangian is the difference between a system's kinetic and potential energy. The function is a useful quantity when considering the formulation of the equation of motion, and is given by.

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{3.37}$$

Where:

$\mathcal{L}$	-	Lagrangian	$[\frac{kgm^2}{s^2}]$
$\mathcal{K}$	-	Kinetic energy	$[\frac{kgm^2}{s^2}]$
$\mathcal{P}$	-	Potential energy	$[\frac{kgm^2}{s^2}]$

The kinetic energy ($\mathcal{K}$) of the simple pendulum system can be derived as follows.

$$\begin{aligned}\mathcal{K} &= \frac{1}{2}m_p\dot{P}_p^2 \\ &= \frac{1}{2}m_p\dot{x}_t^2 + \frac{1}{2}m_p\dot{z}_t^2 + m_p\dot{x}_t\dot{L}_w\sin\phi - m_p\dot{z}_t\dot{L}_w\cos\phi \\ &\quad + m_p\dot{x}_t\dot{\phi}L_w\cos\phi + m_p\dot{z}_t\dot{\phi}L_w\sin\phi \\ &\quad + \frac{1}{2}m_pL_w^2\dot{\phi}^2 + \frac{1}{2}m_p\dot{L}_w^2\end{aligned} \tag{3.38}$$

With the gravity g acting in the opposite direction of the Z-axis (see Fig. 3.6), the potential energy ($\mathcal{P}$) of the simple pendulum system is equal to.

$$\begin{aligned}\mathcal{P} &= m_pgz_p \\ &= m_pgz_t - m_pgL_w\cos\phi\end{aligned} \tag{3.39}$$

Now combining Eq. 3.38 and Eq. 3.38, the extended formulation for the Lagrangian of the simple hanging pendulum is given by.

$$\begin{aligned}\mathcal{L} &= \frac{1}{2}m_p\dot{x}_t^2 + \frac{1}{2}m_p\dot{z}_t^2 + m_p\dot{x}_t\dot{L}_w\sin\phi - m_p\dot{z}_t\dot{L}_w\cos\phi \\ &\quad + m_p\dot{x}_t\dot{\phi}L_w\cos\phi + m_p\dot{z}_t\dot{\phi}L_w\sin\phi \\ &\quad + \frac{1}{2}m_pL_w^2\dot{\phi}^2 + \frac{1}{2}m_p\dot{L}_w^2 \\ &\quad - m_pgz_t + m_pgL_w\cos\phi\end{aligned} \tag{3.40}$$

Equation of Motion

The Euler-Lagrange equation is used to derive the dynamic equation of motion for the simple pendulum system. The Euler-Lagrange equation is defined as [19].

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = \tau_k \tag{3.41}$$

Where:

$\mathcal{L}$	-	Lagrangian	$[\frac{kgm^2}{s^2}]$
q_k	-	Generalized coordinate k	$[-]$
$\dot{q}_k$	-	Derivative of generalized coordinate k	$[-]$
τ_k	-	Generalized force associated with q_k	$[N]$

A few simplifications have been made to reduce the complexity of the Lagrangian (given by Eq. 3.40), these simplifications are based on the following assumptions and considerations.

- Tool-point will only be actuated in the horizontal direction, which leads to $\dot{z}_t = 0$.
- The wire length is considered to have a constant value, hence $\dot{L}_w = 0$.

These assumptions lead to a new simplified expression of the simple pendulum Lagrangian.

$$\begin{aligned}\mathcal{L} = &\frac{1}{2}m_p\dot{x}_t^2 + m_p\dot{x}_t\dot{\phi}L_w\cos\phi + \frac{1}{2}m_pL_w^2\dot{\phi}^2 \\ &- m_pgz_t + m_pgL_w\cos\phi\end{aligned} \tag{3.42}$$

The tool-point is considered to be actuated as pure motion by the industrial robot, and the wire length considered to be kept constant, during this analysis. Leading to the investigation of the force requirements of the tool-point and wire being neglected, and only ϕ will remain as the generalized coordinate of interest (see Eq. 3.32). Hence, the dynamic equation of motion concerning the simple pendulum can be expressed as.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{\phi}} - \frac{\partial \mathcal{L}}{\partial \phi} = 0 \tag{3.43}$$

Inserting Eq. 3.42 into Eq. 3.43, gives the following equation.

$$m_p\ddot{x}_tL_w\cos\phi + m_pL_w^2\ddot{\phi} + m_pgL_w\sin\phi = 0 \tag{3.44}$$

Solving for $\ddot{\phi}$ gives:

$$\ddot{\phi} = -\frac{\cos\phi\ddot{x}_t}{L_w} - \frac{g\sin\phi}{L_w} \tag{3.45}$$

Eq. 3.45, together with Eq. 3.35 and Eq. 3.36 can now be used to describe the motion of the simple pendulum system.

3.2.2 Suspended Load (3D-System)

This subsection will present and derive the formulation of the equation of motion for the suspended load in 3-dimensions. In the same manner, as in the study of the simple pendulum, the suspended payload is considered to be attached to the robot's tool-point via the wire of the winch. The equations of motion are derived from the following assumptions.

- The payload is assumed to be a point mass with a known weight.
- The wire is assumed to be a massless rigid rod.
- The deflection and elongation of the wire is neglected
- The frictional elements between the wire and tool-point are neglected.

An illustration of the 3D suspend load system is shown in Fig. 3.7, this representation is based on the set of Euler-angles, as done by Gustafsson [21]. The suspended load P_p with mass m_p is connected to the tool-point P_t via the wire L_w. When in motion, the load will initiate a rotational angle ϕ_β around the x-axis, and an angle ϕ_α around the ${}^{\phi_\beta}y$-axis. The notation of ${}^{\phi_\beta}y$ and ${}^{\phi_\beta}z$ emphasizes the new coordinate frame created by the first rotation ϕ_β.

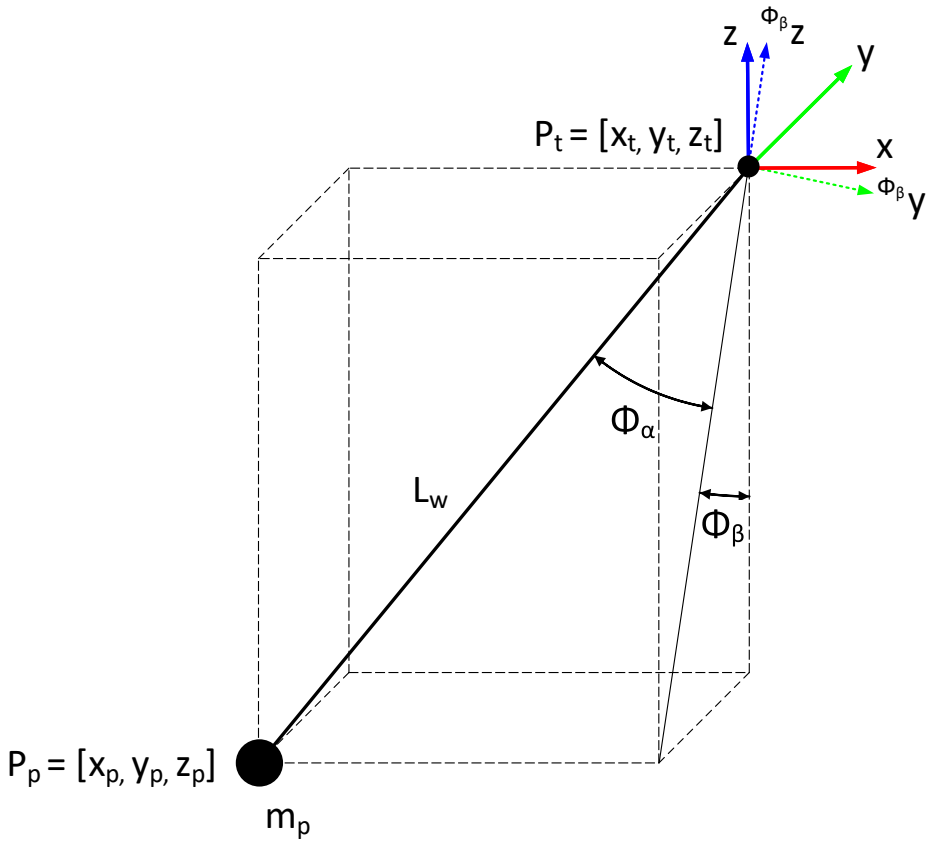


Figure 3.7: An Illustration of the Suspended Load in 3-Dimensions

Payload Position

Using the Euler-angle representation shown in Fig. 3.7, the suspended load position can be expressed as.

$$x_p = x_t - L_w \sin\phi_\alpha \tag{3.46}$$
$$y_p = y_t + L_w \cos\phi_\alpha \sin\phi_\beta \tag{3.47}$$
$$z_p = z_t - L_w \cos\phi_\alpha \cos\phi_\beta \tag{3.48}$$

Combining Eq. 3.46 - Eq. 3.34, the payload position can be rewritten to a vector formulation.

$$P_p = P_t + L_w \begin{bmatrix} -\sin\phi_\alpha \\ \cos\phi_\alpha \sin\phi_\beta \\ -\cos\phi_\alpha \cos\phi_\beta \end{bmatrix} \triangleq P_t + L_w\lambda \tag{3.49}$$

Payload Velocity

The expression for the velocity of the suspended load is derived by differentiating the position expression (Eq. 3.49) and is given by.

$$\dot{P}_p = \dot{P}_t + L_w\dot{\lambda} + \dot{L}_w\lambda \tag{3.50}$$

$$\Rightarrow \begin{bmatrix} \dot{x}_p \\ \dot{y}_p \\ \dot{z}_p \end{bmatrix} = \begin{bmatrix} \dot{x}_t \\ \dot{y}_t \\ \dot{z}_t \end{bmatrix} + L_w \begin{bmatrix} -\dot{\phi}_\alpha \cos\phi_\alpha \\ -\dot{\phi}_\alpha \sin\phi_\alpha \sin\phi_\beta + \dot{\phi}_\beta \cos\phi_\alpha \cos\phi_\beta \\ \dot{\phi}_\alpha \sin\phi_\alpha \cos\phi_\beta + \dot{\phi}_\beta \cos\phi_\alpha \sin\phi_\beta \end{bmatrix} + \dot{L}_w \begin{bmatrix} -\sin\phi_\alpha \\ \cos\phi_\alpha \sin\phi_\beta \\ -\cos\phi_\alpha \cos\phi_\beta \end{bmatrix}$$

Lagrangian

Similar as for the analysis of the simple pendulum, the Lagrangian of the 3-dimensional suspended load is used to formulate the dynamic equation of motion.

Using Eq. 3.50, the kinetic energy of the suspended load can be derived as.

$$\begin{aligned}
\mathcal{K} &= \frac{1}{2} m_p \dot{P}_p^2 \\
&= \frac{1}{2} m_p \dot{x}_t^2 + \frac{1}{2} m_p \dot{y}_t^2 + \frac{1}{2} m_p \dot{z}_t^2 + \frac{1}{2} m_p \dot{L}_w^2 \\
&\quad + \frac{1}{2} m_p \dot{L}_w^2 \cos^2 \phi_\alpha - \frac{1}{2} m_p L_w^2 \cos^2 \phi_\alpha \\
&\quad + \frac{1}{2} m_p L_w^2 \dot{\phi}_\alpha^2 + \frac{1}{2} m_p L_w^2 \dot{\phi}_\beta^2 \cos \phi_\alpha \\
&\quad - m_p L_w \dot{x}_t \sin \phi_\alpha + \frac{1}{2} m_p L_w^2 \dot{\phi}_\alpha \sin^2 \phi_\alpha \\
&\quad - m_p L_w \dot{\phi}_\alpha \dot{x}_t \cos \phi_\alpha - \frac{1}{2} m_p L_w \dot{L}_w \dot{\phi}_\alpha \sin^2 \phi_\alpha \\
&\quad - m_p \dot{L}_w \dot{z}_t \cos \phi_\alpha \cos \phi_\beta + m_p \dot{L}_w \dot{y}_t \cos \phi_\alpha \sin \phi_\beta \\
&\quad + m_p L_w \dot{\phi}_\beta \dot{y}_t \cos \phi_\alpha \cos \phi_\beta + m_p L_w \dot{\phi}_\alpha \dot{z}_t \cos \phi_\beta \sin \phi_\alpha \\
&\quad + m_p L_w \dot{\phi}_\beta \dot{z}_t \cos \phi_\alpha \sin \phi_\beta - m_p L_w \dot{\phi}_\alpha \dot{y}_t sin \phi_\alpha \sin \phi_\beta
\end{aligned} \tag{3.51}$$

Defining gravity in the opposite direction of the z-axis (see Fig. 3.7), the potential energy of the 3D suspended load is given by.

$$\begin{aligned}
\mathcal{P} &= m_p g P_p \\
&= m_p g z_t - m_p g L_w \cos \phi_\alpha \cos \phi_\beta
\end{aligned} \tag{3.52}$$

The Lagrangian for the 3D suspended load can be expressed by Eq. 3.53, where $\mathcal{K}$ and $\mathcal{P}$ is equal to Eq. 3.51 and Eq. 3.52, respectively.

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{3.53}$$

Equation of Motion

Analogous to the system of the simple pendulum, the Euler-Lagrange equation defined by Eq. 3.41 is used to find the dynamic equation of motion for the 3-dimensional suspended load. The generalized coordinate vector will be composed of the two Euler-angles ϕ_α and ϕ_β, and the associated generalized force will equal $\tau_k = 0$. The Euler-Lagrange equation for the suspended load is given by.

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_k} - \frac{\partial \mathcal{L}}{\partial q_k} = 0 \tag{3.54}$$

Where the generalized coordinate vector is given by

$$q_k = \begin{bmatrix} \phi_\alpha \\ \phi_\beta \end{bmatrix} \tag{3.55}$$

Solving the Euler-Lagrange equation (Eq. 3.54) by using the Lagrangian derived by Eq. 3.53 and the generalized coordinate vector (Eq. 3.55), gives the following coupled pair of second order differential equations.

$$\begin{aligned} L_w\ddot{\phi}_\alpha = {} & \ddot{x}_t \cos\phi_\alpha + \ddot{y}_t sin\phi_\alpha \sin\phi_\beta \\ & - \ddot{z}_t \sin\phi_\alpha \cos\phi_\beta - g\sin\phi_\alpha\cos\phi_\beta \\ & - 2\dot{L}_w\dot{\phi}_\alpha - L_w\dot{\phi}_\beta^2 \sin\phi_\alpha\cos\phi_\alpha \end{aligned} \tag{3.56}$$

$$\begin{aligned} L_w\cos\phi_\alpha\ddot{\phi}_\beta = {} & -\ddot{y}_t\cos\phi_\beta - \ddot{z}_t\sin\phi_\beta \\ & - g\sin\phi_\beta - 2\dot{L}_w\dot{\phi}_\beta\cos\phi_\alpha \\ & + 2L_w\dot{\phi}_\alpha\dot{\phi}_\beta\sin\phi_\alpha \end{aligned} \tag{3.57}$$

Now, by using the derived equation of position Eq. 3.49 and velocity Eq. 3.50, together with the differential equations Eq. 3.56 and Eq. 3.57 it is possible to describe the dynamic motion of the suspended pendulum.

Throughout this section, Maple was used to in the derivation of the Lagrangian and Euler-Lagrange equations. The Maple script is available in App. B.

3.3 Full System Motion

The experimental setup of the Motion-Lab uses the Stewart platform to simulate the motion of a floating vessel. With the Comau robot mounted on top of the platform, the simulated wave motion will introduce a relative motion of the robot base. Leading to the robot's tool-point and the suspended load to be influenced by both the generated wave motion and the actuation of the robot joints.

In the previous sections, expressions for the robot kinematics (Sec. 3.1) and the suspended load motion (Sec. 3.2) are derived relative to their internal frame coordinate. This section will expand these expressions, and formulate the relative motion and relation between the moving frames of the full system.

An illustration of the full system setup is shown in Fig. 3.8. This figure shows the body-fixed coordinate systems of the equipment, together with the homogeneous transformation between them. A world coordinate is added to act as a global reference to the motion. See Tab. 3.3 for a detailed description of the coordinate system annotations.

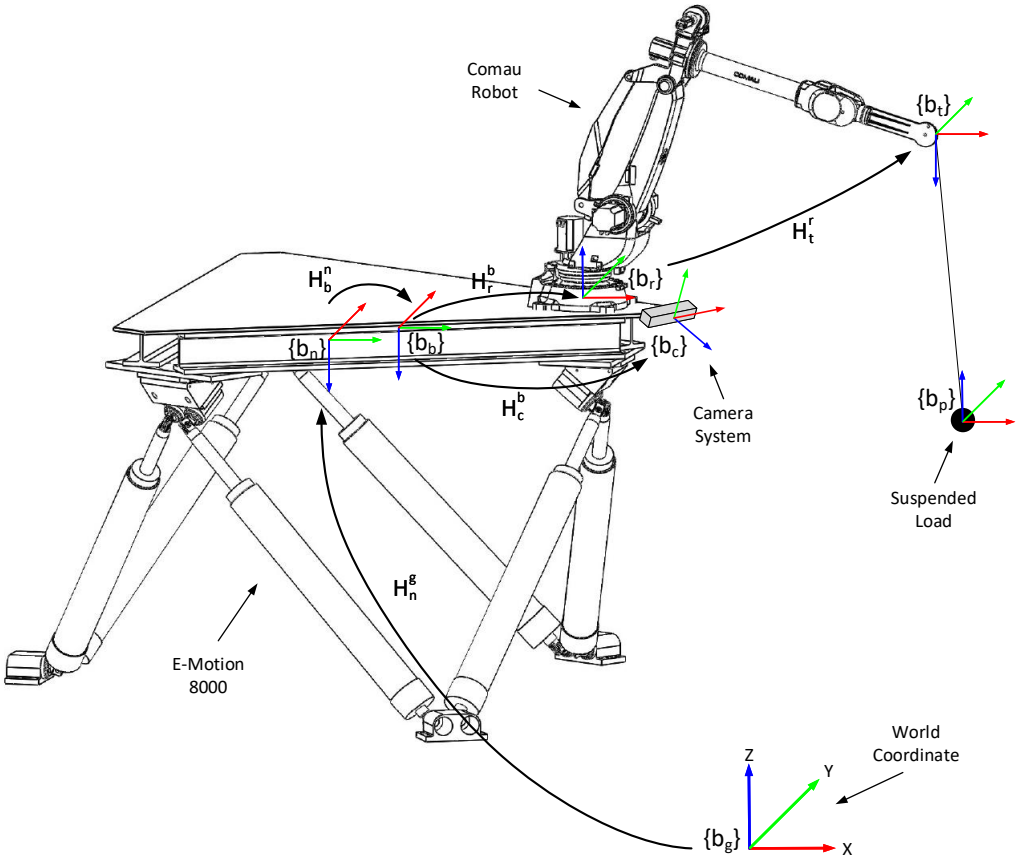


Figure 3.8: An Overview of the Full System Setup, with Body-Fixed Coordinate System and Transformations

Table 3.3: Coordinate System Annotations used for the Full System Kinematics

Annotation	Description
$\{b_g\}$	World Coordinate
$\{b_n\}$	E-Motion 8000 (Neutral)
$\{b_b\}$	E-Motion 8000 (Body-fixed)
$\{b_r\}$	Comau Robot (Base)
$\{b_t\}$	Comau Robot (Tool-point)
$\{b_c\}$	Camera System
$\{b_p\}$	Suspended Payload

As shown in the illustration, the E-Motion 8000 has been assigned with two coordinate system ($\{b_n\}$ and $\{b_b\}$), the former dictates the coordinate system of the platform in neutral position, i.e. in the position when the platform is not exposed to a wave-induced motion. The latter describes the coordinate system for when the platform is in motion. This notation is introduced to easier describe the homogeneous transformations between the platform and the equipment installed on the platform.

Homogeneous Transformations

Homogeneous transformations will be used to describe the relative motion between the equipment in focus. The world coordinate $\{b_g\}$ will be used as a general reference to the different frames illustrated in Fig. 3.8. The primary transformations are shown in the overview, where the superscripts and subscripts referrers to the relative frame and the focus frame, respectively. E.g. H_r^b is the homogeneous transformation of the Comau robot base-frame $\{b_r\}$ given in the E-Motion 8000 body-fixed frame $\{b_b\}$.

3.3.1 Calibrated Transformations

Some of the presented homogeneous transformations remain as fixed relations during the wave-induced simulation, i.e., these transformations will persist as constant relationships independent of the system motion. Which is the case for the homogeneous transformations related to the equipment installed on the Stewart platform, as well as the transformation between the world coordinate and the neutral frame of the Stewart platform. These transformations are obtainable through calibration. Tørdal et al. [22] introduces a method to obtain these calibrations, where the results of this research are used in this thesis. Tab. 3.4 lists the fixed homogeneous transformation obtainable with calibration.

Table 3.4: Fixed Homogeneous Transformations Obtained from Calibration

Annotation	Description
H_n^g	E-Motion 8000 neutral-frame relative to World Coordinate
H_r^b	Comau Robot base-frame relative to EM 8000 body-fixed frame
H_c^b	Kinect V2 Camera relative to EM 8000 body-fixed frame

3.3.2 Stewart Platform Motion

Similar to a vessel exposed to waves at sea, the wave motion simulated by the Stewart platform is a 6-DOF motion. As depicted in Fig. 3.9, the sway (x), surge (y) and heave (z) are notations for the position, and roll, pitch and yaw are denoted as the rotational angles ϕ, θ and ψ, respectively.

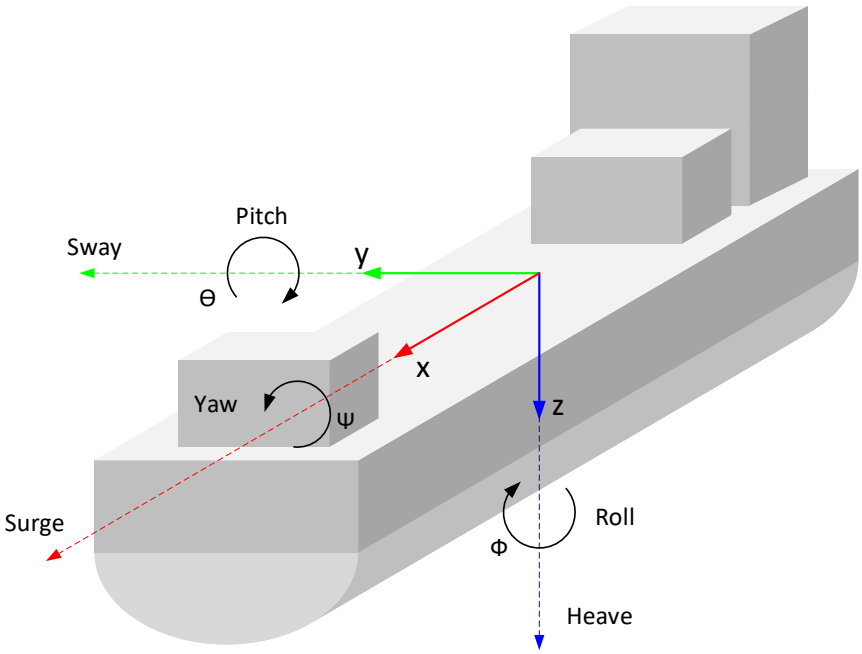


Figure 3.9: Definition of Axis and Orientation for Vessel Motion

Relating the annotation of Fig. 3.9 to the configuration of the Motion-Lab in Fig. 3.8. The sway, surge and heave can be defined as a vector P_b^n, which describes the translational position of the frame $\{b_b\}$ given in $\{b_n\}$, where $\{b_n\}$ is defined as the static frame of the Stewart platform, and $\{b_b\}$ is the frame induced by wave motion.

$$P_b^n = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3.58}$$

Where:

x	-	Sway	$[m]$
y	-	Surge	$[m]$
z	-	Heave	$[m]$

The roll, pitch, and yaw can be defined as a vector α.

$$\alpha = \begin{bmatrix} \phi \\ \theta \\ \psi \end{bmatrix} \tag{3.59}$$

Where:

$$\begin{array}{lcll} \phi & - & \text{Roll} & [m] \\ \theta & - & \text{Pitch} & [m] \\ \psi & - & \text{Yaw} & [m] \end{array}$$

The relative orientation of frame $\{b_b\}$ given in $\{b_n\}$ can be expressed as a successive sequence of rotations.

$$\begin{aligned} R_b^n(\alpha) &= R_z(\phi)R_y(\theta)R_x(\psi) \\ &= \begin{bmatrix} \cos\phi & -\sin\phi & 0 \\ \sin\phi & \cos\phi & 0 \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} \cos\theta & 0 & \sin\theta \\ 0 & 1 & 0 \\ -\sin\theta & 0 & \cos\theta \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos\psi & -\sin\psi \\ 0 & \sin\psi & \cos\psi \end{bmatrix} \\ &= \begin{bmatrix} \cos\phi\cos\theta & -\sin\phi\cos\psi + \cos\phi\sin\theta\sin\psi & \sin\phi\sin\psi + \cos\phi\sin\theta\cos\psi \\ \sin\phi\cos\theta & \cos\phi\cos\psi + \sin\phi\sin\theta\sin\psi & -\cos\phi\sin\psi + \sin\phi\sin\theta\cos\psi \\ -\sin\theta & \cos\theta\sin\psi & \cos\theta\cos\psi \end{bmatrix} \end{aligned} \tag{3.60}$$

A new vector η is introduced, which is a collected vector of the sway, surge, heave vector P_b^n (Eq. 3.58) and the roll, pitch, yaw vector α (Eq. 3.59).

$$\eta = \begin{bmatrix} P_n^b \\ \alpha \end{bmatrix} = \begin{bmatrix} x \\ y \\ z \\ \phi \\ \theta \\ \psi \end{bmatrix} \tag{3.61}$$

The collected vector η, can now be used to describe the Stewart platform relative translation and rotation of frame $\{b_b\}$ given in $\{b_n\}$. The orientation vector η, together with it's respective time derivative $\dot{\eta}$ (velocity) and second time derivative $\ddot{\eta}$ (acceleration) are characterized as the governing parameters for the motion of the Stewart platform.

Stewart Platform Position and Orientation

The relative position and orientation of the Stewart platform in motion can derived as the homogeneous transformation H_b^n (transformation of the E-Motion 8000 body-fixed frame, given in the E-Motion 8000 neutral-frame), which is formulated by using Eq. 3.58 and Eq. 3.60.

$$H_b^n = \begin{bmatrix} R_b^n(\alpha) & P_b^n \\ 0 & 1 \end{bmatrix} \tag{3.62}$$

Using the obtained calibration data H_n^g, the position and orientation of the Stewart platform relative to the world coordinate frame $\{b_g\}$ is given by.

$$H_b^g = H_n^g H_b^n \tag{3.63}$$

Stewart Platform Velocity

The wave-induced motion simulated by the Stewart platform introduces translational and rotational velocities to the equipment installed on the platform. It is therefore desired to formulate an expression which describes the angular velocity relative to the platform's body-fixed frame $\{b_b\}$.

The time derivative of the previous defined η (Eq. 3.61) is equal to.

$$\dot{\eta} = \begin{bmatrix} \dot{P}_b^n \\ \dot{\alpha} \end{bmatrix} \tag{3.64}$$

The vector $\omega_{n,b}^b$ denotes the angular velocity corresponding to the time derivative of the rotation matrix R_b^n expressed relative to the body-fixed frame $\{b_b\}$. The relation between this angular velocity vector and $\dot{\alpha}$ can be found by the following equation.

$$\dot{\eta} = J(\eta)v \tag{3.65}$$

Where v is defined as.

$$v = \begin{bmatrix} \dot{P}_b^n \\ \omega_{n,b}^b \end{bmatrix} \tag{3.66}$$

$J(\eta)$ is referred to as the ship Jacobian and is expressed as.

$$J(\eta) = \begin{bmatrix} I & 0 \\ 0 & T(\alpha) \end{bmatrix} \tag{3.67}$$

Where I is the identity matrix, and $T(\alpha)$ is a transformation matrix equal to.

$$T(\alpha) = \begin{bmatrix} 1 & \frac{(\sin\phi\sin\theta)}{\cos\theta} & \frac{(\cos\phi\sin\theta)}{\cos\theta} \\ 0 & \cos\phi & -\sin\phi \\ 0 & \frac{(\sin\phi)}{\cos\theta} & \frac{(\cos\phi)}{\cos\theta} \end{bmatrix} \tag{3.68}$$

Using these formulations, Eq. 3.65 can be rewritten, and solved for the vector v, which contains the translational velocity $\dot{p}_b^n$, and the angular velocity vector $\omega_{n,b}^b$.

$$v = \begin{bmatrix} \dot{P}_b^n \\ \omega_{n,b}^b \end{bmatrix} = J(\eta)^{-1}\dot{\eta} \tag{3.69}$$

The notion of skew matrices is introduced to aid in the upcoming derivations. Where the skew matrix of an arbitrary vector $\beta = [\beta_x, \beta_y, \beta_z]^T$ is given by.

$$S(\beta) = \begin{bmatrix} 0 & -\beta_z & \beta_y \\ \beta_z & 0 & -\beta_x \\ -\beta_y & \beta_x & 0 \end{bmatrix} \tag{3.70}$$

The rotational velocity of the Stewart platform can now be expressed as the derivative of the rotation matrix R_b^n, which is derived by using the skew matrix of the body fixed velocity vector $\omega_{n,b}^b$.

$$\dot{R}_b^n = R_b^n S(\omega_{n,b}^b) \tag{3.71}$$

Stewart Platform Acceleration

In the same manner, as the velocities, the wave motion will induce translational and rotational accelerations to the equipment installed on the Stewart platform.

The same procedure as for the velocity formulation will be performed. The second time derivative of Eq. 3.61 is defined as.

$$\ddot{\eta} = \begin{bmatrix} \ddot{P}_b^n \\ \ddot{\alpha} \end{bmatrix} \tag{3.72}$$

The time derivative of Eq. 3.65 equals.

$$\ddot{\eta} = \dot{J}(\eta)v + J(\eta)\dot{v} \tag{3.73}$$

Where $\dot{J}$ can be derived as.

$$\dot{J}(\eta) = \begin{bmatrix} \frac{\partial(J(\eta)\dot{\eta}_1)}{\partial\eta_1} & \cdots & \frac{\partial(J(\eta)\dot{\eta}_1)}{\partial\eta_6} \\ \vdots & \ddots & \vdots \\ \frac{\partial(J(\eta)\dot{\eta}_6)}{\partial\eta_1} & \cdots & \frac{\partial(J(\eta)\dot{\eta}_6)}{\partial\eta_6} \end{bmatrix} \tag{3.74}$$

To obtain an expression for $\dot{\omega}^b_{n,b}$, Eq. 3.73 is rewritten and solved for $\dot{v}$, which equals

$$\dot{v} = \begin{bmatrix} \dot{P}^n_b \\ \dot{\omega}^b_{n,b} \end{bmatrix} = J(\eta)^{-1}(\ddot{\eta} - \dot{J}(\eta)\dot{\eta}) \tag{3.75}$$

The rotational acceleration of the Stewart platform can now be expressed as the time derivative of Eq. 3.71, which equals.

$$\begin{aligned} \ddot{R}^n_b &= \dot{R}^n_b S(\omega^b_{n,b}) + R^n_b S(\dot{\omega}^b_{n,b}) \\ &= R^n_b S(\omega^b_{n,b}) S(\omega^b_{n,b}) + R^n_b S(\dot{\omega}^b_{n,b}) \end{aligned} \tag{3.76}$$

3.3.3 Updated Robot Tool-Point Motion

With the motion induced by the Stewart platform, the base-frame of the robot will experience a motion relative to the world coordinate. As mentioned earlier, the expressions describing the robot's tool-point (Sec. 3.1.2) are derived relative to the internal frame coordinate, which corresponds to the base-frame $\{b_r\}$ of Fig. 3.8. Meaning that the equations of the tool-point motion need to be updated for Stewart platform motion.

The available calibration matrix for the homogeneous transformation H^b_r describes the position and orientation of the Comau robot base-frame $\{b_r\}$ relative to the Stewart platform's body-fixed frame $\{b_b\}$ and is given on the form.

$$H^b_r = \begin{bmatrix} R^b_r & P^b_r \\ 0 & 1 \end{bmatrix} \tag{3.77}$$

Similar, the calibration matrix for the homogeneous transformation H^g_n, which describes the position and orientation of the Stewart platform's neutral frame $\{b_n\}$ relative to the world coordinate $\{b_g\}$, is given by.

$$H^g_n = \begin{bmatrix} R^g_n & P^g_n \\ 0 & 1 \end{bmatrix} \tag{3.78}$$

Tool-Point Position

The robot's tool-point position relative to the Stewart platform's neutral-frame $\{b_n\}$ can be formulated as.

$$P^n_t = P^n_b + R^n_b(P^b_r + R^b_r P_t) \tag{3.79}$$

Where P_b^n and R_b^n are the Stewart platform position and orientation, given by Eq. 3.58 and Eq. 3.60, respectively. P_t is obtained from Eq. 3.8, which describes the tool-point position relative to the internal frame of the robot.

Using Eq. 3.78 and Eq. 3.79, the position of the robot's tool-point relative to the world coordinate can be derived as.

$$P_t^g = P_n^g + R_n^g P_t^n \tag{3.80}$$

Tool-Point Velocity

The expression of the tool-point velocity relative to the world coordinate frame $\{b_g\}$ is simply the time derivative of Eq. 3.79, this is due to the calibrated homogeneous transformation H_n^g being a constant relation. Hence, the velocity of the robot's tool-point is given by.

$$\dot{P}_t^g = \dot{P}_t^n = \dot{P}_b^n + \dot{R}_b^n(P_r^b + R_r^b P_t) + R_b^n(R_r^b \dot{P}_t) \tag{3.81}$$

Tool-Point Acceleration

The acceleration of the robot's tool-point equals the time derivative of Eq. 3.81, which is derived as.

$$\begin{aligned}\ddot{P}_t^g = \ddot{P}_t^n = \ddot{P}_b^n + \ddot{R}_b^n(P_r^b + R_r^b P_t) \\ + 2\dot{R}_b^n(R_r^b \dot{P}_t) + R_b^n(R_r^b \ddot{P}_t)\end{aligned} \tag{3.82}$$

3.3.4 Updated Suspended Load Motion

The expressions describing the motion of the robot's tool-point (Sec. 3.3.3) are now updated to include the relative motion of the Stewart platform. The same operation is needed for the motion of the suspended load, which is derived relative to the tool-point (Sec. 3.2).

The change of introducing the relative motion of the Stewart platform to the suspended load is a relative easy modification. The new expressions will simply use the updated expressions of the tool-point motion (Eq. 3.80, Eq. 3.81 and Eq. 3.82).

Suspended Load Position

The updated expression of the suspended load position, previously derived as Eq. 3.49, is given by.

$$P_p^g = P_t^g + L_w \lambda \tag{3.83}$$

Suspended Load Velocity

The same goes for the expression describing the suspended load velocity, which is previously given by Eq. 3.50, is now given by.

$$\dot{P}_p^g = \dot{P}_t^g + L_w\dot{\lambda} + \dot{L}_w\lambda \tag{3.84}$$

Suspended Load Equation of Motion

The expressions describing the equation of motion for the suspended load (previously given by Eq. 3.56 and Eq. 3.57), can be updated for platform motion by introducing the new expression of the tool-point acceleration (Eq. 3.82). The components of the tool-point's acceleration vector equals

$$\ddot{P}_t^g = \begin{bmatrix} \ddot{x}_t^g \\ \ddot{y}_t^g \\ \ddot{z}_t^g \end{bmatrix} \tag{3.85}$$

Which yields the following updated expressions for the suspended load's equation of motion.

$$\begin{aligned} L_w\ddot{\phi}_\alpha = {} & \ddot{x}_t^g \cos\phi_\alpha + \ddot{y}_t^g sin\phi_\alpha \sin\phi_\beta \\ & - \ddot{z}_t^g \sin\phi_\alpha \cos\phi_\beta - g\sin\phi_\alpha\cos\phi_\beta \\ & - 2\dot{L}_w\dot{\phi}_\alpha - L_w\dot{\phi}_\beta^2 \sin\phi_\alpha\cos\phi_\alpha \end{aligned} \tag{3.86}$$

$$\begin{aligned} L_w\cos\phi_\alpha\ddot{\phi}_\beta = {} & -\ddot{y}_t^g\cos\phi_\beta - \ddot{z}_t^g\sin\phi_\beta \\ & - g\sin\phi_\beta - 2\dot{L}_w\dot{\phi}_\beta\cos\phi_\alpha \\ & + 2L_w\dot{\phi}_\alpha\dot{\phi}_\beta\sin\phi_\alpha \end{aligned} \tag{3.87}$$

3.3.5 Camera System

A camera system, capable of tracking the motion of the suspended load, is considered to be installed and available in the motion-lab. This system can either be a camera installed on the platform as shown in Fig. 3.8, or by making use of the motion-capture system, described in Sec. 2.1.5.

For this type of system, the suspended load's position will be measured relative to the camera frame $\{b_c\}$. It would preferably be desired to have these measurements given relative to the Stewart platform, hence a calibration of the homogeneous transformation between the camera and the body-fixed frame of the Stewart platform $\{b_b\}$ is needed, where this transformation is defined as H_c^b. The suspended load's position relative to the body-fixed frame of the Stewart platform can then be described by Eq. 3.88, where P_c^b and R_c^b are the translation vector and rotation matrix, respectively, of the calibrated transformation matrix H_c^b.

$$P_p^b = P_c^b + R_c^b P_p^c \tag{3.88}$$

For the future development of the control systems, a measurement of the pendulum Euler angles will be required. These measurements are considered to be obtainable by altering the describing equations Eq. 3.46 and Eq. 3.47 into the following form.

$$\begin{aligned} x_p^b &= x_t^b - L_w \sin\phi_\alpha \\ \Rightarrow \phi_\alpha &= \arcsin\big(\frac{x_t^b - x_p^b}{L_w}\big) \end{aligned} \tag{3.89}$$

$$\begin{aligned} y_p &= y_t + L_w \cos\phi_\alpha \sin\phi_\beta \\ \Rightarrow \phi_\beta &= \arcsin\big(\frac{y_p^b - y_t^b}{L_w \cos\phi_\alpha}\big) \end{aligned} \tag{3.90}$$

3.4 Control System Design

One of the main tasks in this thesis is to design and develop a controller capable of stabilizing the suspended load in sideways motion, i.e., the suspended load's sway and surge motion (see Fig. 3.9) should be minimized when the system is exposed to a wave-induced motion. This section will present the theory behind the general state-space representation of a dynamic system, together with the control law and state-feedback design. State-space linearization and estimator design will also be covered, due to the state-estimator dependency of the dynamic non-linear system of the suspended load.

3.4.1 State-Space Modelling

State-space representation of a dynamic system is based on the idea of state-variables to describe the differential equations of a system. Ordinary differential equations (ODEs) which describe a dynamic system can be formulated as a set of first-order ODEs in the vector-valued state of the system, where the solution is visualized as the trajectory of this state-vector in space [23].

A state-space system is described by two governing equations, given by Eq. 3.91 and Eq. 3.92. A block diagram of the general state-space system is illustrated by Fig. 3.10.

$$\dot{x} = Ax + Bu \tag{3.91}$$
$$y = Cx + Du \tag{3.92}$$

Where:

x	-	State vector	$x(t) \in \mathbb{R}^n$
u	-	Input vector	$u(t) \in \mathbb{R}^r$
y	-	Output vector	$y(t) \in \mathbb{R}^m$
A	-	System matrix	$A \in \mathbb{R}^{n \times n}$
B	-	Input matrix	$B \in \mathbb{R}^{n \times r}$
C	-	Output matrix	$C \in \mathbb{R}^{m \times n}$
D	-	Direct feed-through term	$D \in \mathbb{R}^{m \times r}$

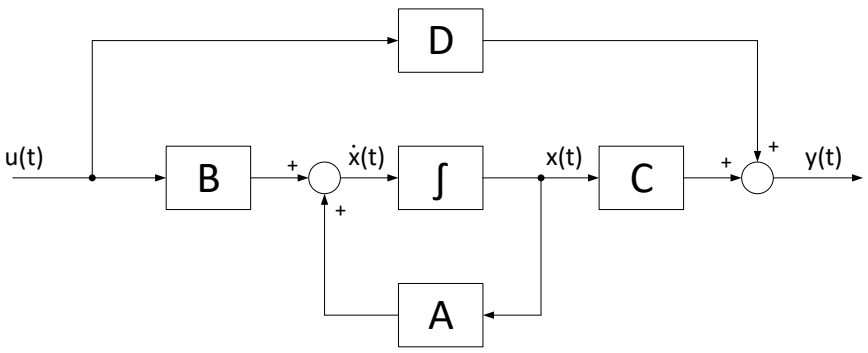


Figure 3.10: State-Space Represented with Block Diagram

3.4.2 State-Space Linearization

Concerning systems with smooth non-linearities and continuous derivatives, it is possible to formulate an approximate linear model which is valid around a definite operation point, this approach is known as the small-signal method [23].

Considering the non-linear system in a state-variable form, the system can be expressed as n first-order differential equations, where the state of the system is described as [24]:

$$\begin{aligned} \dot{x}_1 &= f_1(x_1 \dots x_n, u_1 \dots u_r) \\ \dot{x}_2 &= f_2(x_1 \dots x_n, u_1 \dots u_r) \\ &\dots \\ \dot{x}_n &= f_n(x_1 \dots x_n, u_1 \dots u_r) \end{aligned} \tag{3.93}$$

and the system output is described by.

$$\begin{aligned} y_1 &= h_1(x_1 \dots x_n, u_1 \dots u_r) \\ y_2 &= h_2(x_1 \dots, x_n, u_1 \dots u_r) \\ &\dots \\ y_m &= h_m(x_1 \dots x_n, u_1 \dots u_r) \end{aligned} \tag{3.94}$$

With $x = [x_1 \dots x_n]^\top$, $u = [u_1 \dots u_r]^\top$, and $y = [y_1 \dots y_m]^\top$, the describing equations of the non-linear system can be given by Eq. 3.95 and Eq. 3.96

$$\dot{x} = f(x, u) \tag{3.95}$$

$$y = h(x, u) \tag{3.96}$$

where f and h are non-linear functions of the system states x and the input u.

$$f(x, u) = \begin{bmatrix} f_1(x, u) \\ f_2(x, u) \\ \vdots \\ f_n(x, u) \end{bmatrix} \tag{3.97}$$

$$h(x, u) = \begin{bmatrix} h_1(x, u) \\ h_2(x, u) \\ \vdots \\ h_m(x, u) \end{bmatrix} \tag{3.98}$$

Equilibrium Point

The linearization is conducted by approximating a linear model to fit the non-linear system around an equilibrium point. The equilibrium values of x_0 and u_0 is chosen such that $\dot{x}_0 = f(x_0, u_0) = 0$. Further, new coordinates Δx, Δy, Δu are denoted as [25].

$$\Delta x = x - x_0 \tag{3.99}$$
$$\Delta y = y - h(x_0, u_0) \tag{3.100}$$
$$\Delta u = u - u_0 \tag{3.101}$$

Which leads to the linearized model of the non-linear system (Eq. 3.95 and Eq. 3.96) around the equilibrium point x_0 and u_0 is given by.

$$\Delta \dot{x} = A\Delta x + B\Delta u \tag{3.102}$$
$$\Delta y = C\Delta x + D\Delta u \tag{3.103}$$

A, B, C, and D are new state-space matrices evaluated at x_0 and u_0 and are derived as follows.

$$A = \left[\frac{\partial f(x,u)}{\partial x}\right]_{x_0,u_0} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial f_1}{\partial x_n}\Big|_{x_0,u_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial f_n}{\partial x_n}\Big|_{x_0,u_0} \end{bmatrix} \tag{3.104}$$

$$B = \left[\frac{\partial f(x,u)}{\partial u}\right]_{x_0,u_0} = \begin{bmatrix} \frac{\partial f_1}{\partial u_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial f_1}{\partial u_r}\Big|_{x_0,u_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial u_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial f_n}{\partial u_r}\Big|_{x_0,u_0} \end{bmatrix} \tag{3.105}$$

$$C = \left[\frac{\partial h(x,u)}{\partial x}\right]_{x_0,u_0} = \begin{bmatrix} \frac{\partial h_1}{\partial x_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial h_1}{\partial x_n}\Big|_{x_0,u_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial x_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial h_m}{\partial x_n}\Big|_{x_0,u_0} \end{bmatrix} \tag{3.106}$$

$$D = \left[\frac{\partial h(x,u)}{\partial u}\right]_{x_0,u_0} = \begin{bmatrix} \frac{\partial h_1}{\partial u_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial h_1}{\partial u_r}\Big|_{x_0,u_0} \\ \vdots & \ddots & \vdots \\ \frac{\partial h_m}{\partial u_1}\Big|_{x_0,u_0} & \cdots & \frac{\partial h_m}{\partial u_r}\Big|_{x_0,u_0} \end{bmatrix} \tag{3.107}$$

3.4.3 State-Space Control

Advantages of state-space control compared to classical control design (transfer-function-based methods), is the available technique where dynamic compensations can be design based on working directly with the systems state-variables. An additional feature of the state-space control is the relative simplicity of how to treat multi-input multi-output (MIMO) systems.

The state-space control design method can be described as a sequence of four independent steps [23].

- **Finding the control law:** This involves the selection of a set of pole-locations for the closed-loop system, such that the dynamic response corresponds to the desired characteristics. Here, an assumption is taken that all elements of the state-vector are available.
- **Estimator Design:** More often than not, a full state-feedback for a system is not available. An estimator allows for an estimation of the entire state-vector based on available system measurements.
- **Combining the control law and estimator:** The control law with its full state-feedback dependency, is combined with the designed estimator.
- **Reference tracking:** Introduce a reference tracking control to the system, such that the plant will track external command signals with a satisfactory response.

3.4.4 State-Feedback Design

A dynamic system of n-th order has a total of n roots which determines the eigenbehavior of the system. The dynamics of these n states can be changed to an appropriate system response by modification of the location of the n roots. In state-space control design, it is desired to find the control law as a feedback of the linear combination of the systems state-variables. The control law is given by.

$$u = -Kx = -\begin{bmatrix} K_1 & K_2 & \dots & K_n \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_n \end{bmatrix} \tag{3.108}$$

In the state-feedback control design, an assumption is made that the full-state feedback of the system state-vector x is available. In practice, this is not always true, but for the feedback purpose, this assumption is made to proceed with the control design. For an n-th order system, the feedback gain K will be of dimension $(n \times 1)$. Satisfactory system response, is determined by the root location, manipulation of these roots are achievable by appropriate selection of the gain values in K.

A block diagram representation of the state-feedback design is illustrated by Fig. 3.11. For a pure state-feedback control the input signal is considered to be equal to zero $u(t) = 0$.

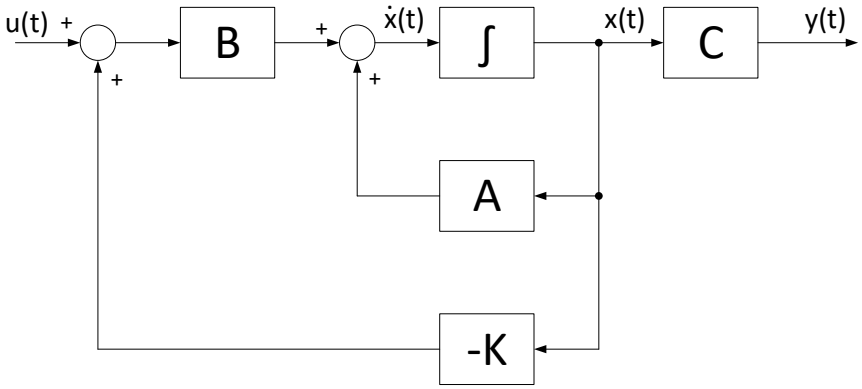


Figure 3.11: Block Diagram of a State-Feedback System

Inserting the feedback law of Eq. 3.108, into the general state-space system described by Eq. 3.91, gives the following state equation.

$$\dot{x} = Ax - BKx \tag{3.109}$$

The characteristic equation of the closed-loop system can be derived as.

$$det[sI - (A - BK)] = 0 \tag{3.110}$$

Evaluating Eq. 3.110, gives an n-th order polynomial in the s-domain, consisting of the feedback gains $K_1, K_2, \ldots, K_n$. Desired root locations can be achieved by assigning appropriate values to these gains. There exist several methods for selecting appropriate values of K, where the manual approach is to match the values of the feedback gains with a set of desired root locations such as

$$s = s_1, s_2, \ldots, s_n \tag{3.111}$$

The corresponding characteristic equation of the desired roots (characteristic control equation) can be expressed as.

$$\alpha_c = (s - s_1)(s - s_2)\ldots(s - s_n) \tag{3.112}$$

Appropriate values of K can now be found by matching the characteristic equation of the closed-loop system in Eq. 3.110, with the characteristic control equation in Eq. 3.112.

3.4.5 Linear Quadratic Regulator (LQR)

Opposed to the methods of manual pole placement, an effective and widely used technique in system control design is the optimal linear quadratic regulator (LQR). The LQR can be considered as an optimization problem, where the aim is to minimize the quadratic cost function.

For a general state-space system given by Eq. 3.95, the quadratic cost function is given by.

$$\mathcal{J} = \int_0^\infty \left(x^\top Q x + u^\top R u \right) \tag{3.113}$$

The control law that minimizes the cost function is given by the state feedback equation of Eq. 3.108, where Q and R are assigned weighting matrices. The solution of the minimization problem can found by the Riccati method, where the Riccati equation is expressed as.

$$A^\top P + PA - PBR^{-1}B^\top P + Q = 0 \tag{3.114}$$

Isolating and solving for P, the optimal feedback gain can be calculated by.

$$K = R^{-1}B^\top P \tag{3.115}$$

The weighting matrices Q and R are often specified as diagonal matrices, where Q penalizes the system states, and R penalizes the control effort. By modifying these matrices, the designer can alter the trade-off between performance and control effort, to achieve an acceptable outcome. As initial starting values to the LQR design, Bryson's Rule is applicable [23], where the diagonal elements of Q and R are assigned values based on the acceptable values of x and u.

$$Q_{ii} = 1/\text{maximum acceptable value of } [x_i^2] \tag{3.116}$$

$$R_{ii} = 1/\text{maximum acceptable value of } [u_i^2] \tag{3.117}$$

Additional influence of the weight matrices can be listed as.

- **Increasing all** Q_{ii} gives faster total system dynamics, but will require higher control values.
- **Increasing all** R_{ii} suppresses the required magnitude of the control values.
- **Increasing a element of** Q_{ii} gives faster eigenbehavior of the corresponding system state.

3.4.6 Reference Input (Pre-filter)

A state-feedback control as described in Sec. 3.4.4, can alone not provide a steady-state accuracy. As the reference input varies, it is desired to have the output signal to follow the reference, and this can be achieved by adding a scale $\bar{N}$ (Pre-filter) to the input signal of the existing state-feedback system. The block diagram of such a system is shown in Fig. 3.12.

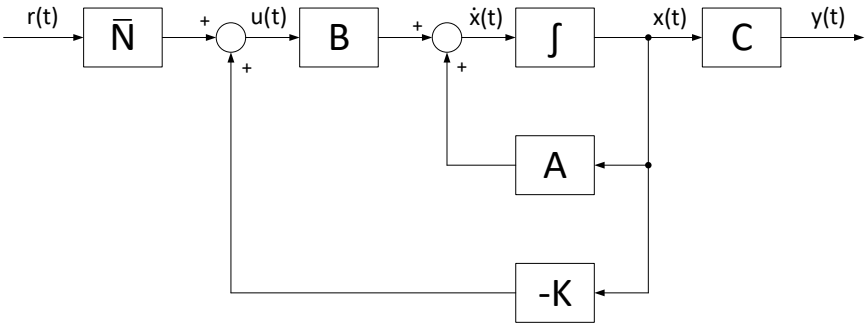


Figure 3.12: Block Diagram of a State-Feedback System with a Pre-Filter

A system with a constant steady-state can be said to have the following requirements; $\dot{x} = 0$ and $y = r$, combining these requirements with the relations derived from the block diagram of Fig. 3.12, gives the following expressions.

$$\dot{x} = (\bar{N}r - Kx)B + Ax = 0 \tag{3.118}$$
$$y = Cx = r \tag{3.119}$$

Expanding Eq. 3.118 and solving for x, gives.

$$x = (BK - A)^{-1}B\bar{N}r \tag{3.120}$$

Inserting Eq. 3.120 into the output equation of Eq. 3.119, gives.

$$y = C\Big[(BK - A)^{-1}B\bar{N}r\Big] = r \tag{3.121}$$

Solving Eq. 3.121 for $\bar{N}$, gives the an expression for deriving the pre-filter gain.

$$\bar{N} = \Big[C(BK - A)^{-1}B\Big]^{-1} \tag{3.122}$$

3.4.7 Integral Control

The pre-filter controller presented in Sec. 3.4.6 will yield in zero steady-state error when exposed to a step command. Unfortunately, this type of controller is not considered to be robust, and if there arise any parameter changes to the system plant, the steady-state error will become non-zero. It is therefore of interest to introduce an integral controller, which is capable of tracking signals which do not go towards zero in steady-state.

A state-space system with an external disturbance d is given by.

$$\dot{x} = Ax + Bu + Gd \tag{3.123}$$
$$y = Cx \tag{3.124}$$

The system output can be designed in a feedback structure with the reference input. Hence the error equals $e = r - y$. An additional integral state z is introduced to the system, and the system error e equals the derivative of this state.

$$\dot{z} = r - Cx = e \tag{3.125}$$
$$z = \int_0^t e\, dt \tag{3.126}$$

The augmented state equations describing the system can be rewritten as.

$$\begin{bmatrix} \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & -C \\ 0 & A \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ B \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r + \begin{bmatrix} 0 \\ G \end{bmatrix} d \tag{3.127}$$
$$y = \begin{bmatrix} 0 & C \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \tag{3.128}$$

The feedback law is given by 3.129, where K_o is the original state-feedback gain and K_e is the feedback gain related to the error states.

$$u = -K_o x + K_e z = -\begin{bmatrix} -K_e & K_o \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \tag{3.129}$$

or simply

$$u = -K \begin{bmatrix} z \\ x \end{bmatrix} \tag{3.130}$$

A block diagram of the presented integral controller is illustrated by Fig. 3.13

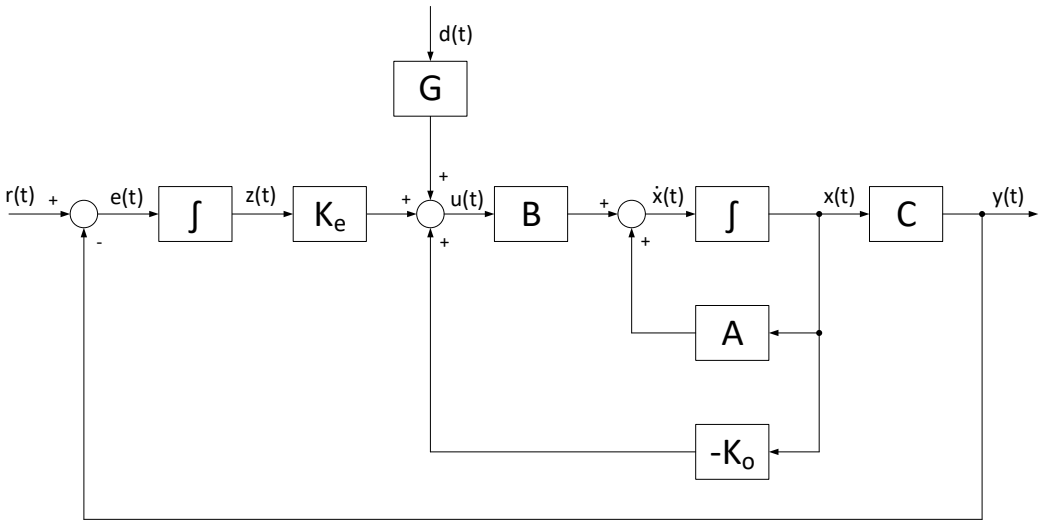


Figure 3.13: Block Diagram of a State-Feedback System with an Integral Control

Assigning the extended state-feedback control gains $K = [-K_e \quad K_o]$ can be conducted by normal methods, such as the LQR presented in Sec. 3.4.5. Part of the augmented state equations of Eq. 3.127 can be formulated as Eq. 3.131, to allow for a LQR optimization.

$$\dot{x_i} = A_i x_i + B_i u \tag{3.131}$$

where

$$x_i = \begin{bmatrix} z \\ x \end{bmatrix} \tag{3.132}$$

$$A_i = \begin{bmatrix} 0 & -C \\ 0 & A \end{bmatrix} \tag{3.133}$$

$$B_i = \begin{bmatrix} 0 \\ B \end{bmatrix} \tag{3.134}$$

3.4.8 Controllability and Oberservability

Controllability refers to the ability of controlling a particular state of a dynamic system by the application of a control input. A system is described to be *fully-controllable* if the system's internal states can be driven into zero state by an external appropriate control input, for any initial state and finite time. To determine if a system plant is fully controllable, the rank of Kalman's controllability matrix needs to be determined. A dynamic system (A,B) of size n is fully controllable if and only if:

$$rank(\mathcal{C}) = rank[B, AB, \ldots, A^{n-1}B] = n \tag{3.135}$$

Observability relates to the concept of deducing information of the internal states of a system, by only monitoring the system's outputs. If every state of the system is observable, the system is said to be *fully-observable*. To determine if a system plant if fully-observable, the rank of Kalman's observability matrix needs to be determined [23]. A dynamic system (A,C) of size n is completely observable if and only if:

$$rank(\wr) = rank[C, CA, \ldots, CA^{n-1}]^\top = n \tag{3.136}$$

3.4.9 Estimator Design

As previously mentioned, the approach of state-feedback design assumes that all system states are available. However, in a physical system, this is rarely the case. Installing sensor equipment to measure all the states of a system can be an expensive and challenging task and in some cases physical impossible. If a system is observable, an estimator can be introduced to the system design, where the states of the system can be reconstructed from the available system output.

Recalling the control law combined with a reference gain as.

$$u = -Kx + \bar{N}r \tag{3.137}$$

A system where not all states are measurable, the full state-vector x is not available. Hence, the estimated state-vector $\hat{x}$ is introduced, which allows for the control law to be derived.

$$u = -K\hat{x} + \bar{N}r \tag{3.138}$$

A common method to estimate the states is to design a full-order model of the plant dynamics. Using the same input u, system matrix A and input vector B as for the plant, a reasonable reconstruction is possible. To assert for small error in the knowledge of the system, a feedback signal is constructed for the difference between the measured output y and estimated output $\hat{y}$, and corrected by a gain L. A general estimator configuration is illustrated by the block diagram in Fig. 3.14, and is given on the form.

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \tag{3.139}$$

$$\hat{y} = C\hat{x} \tag{3.140}$$

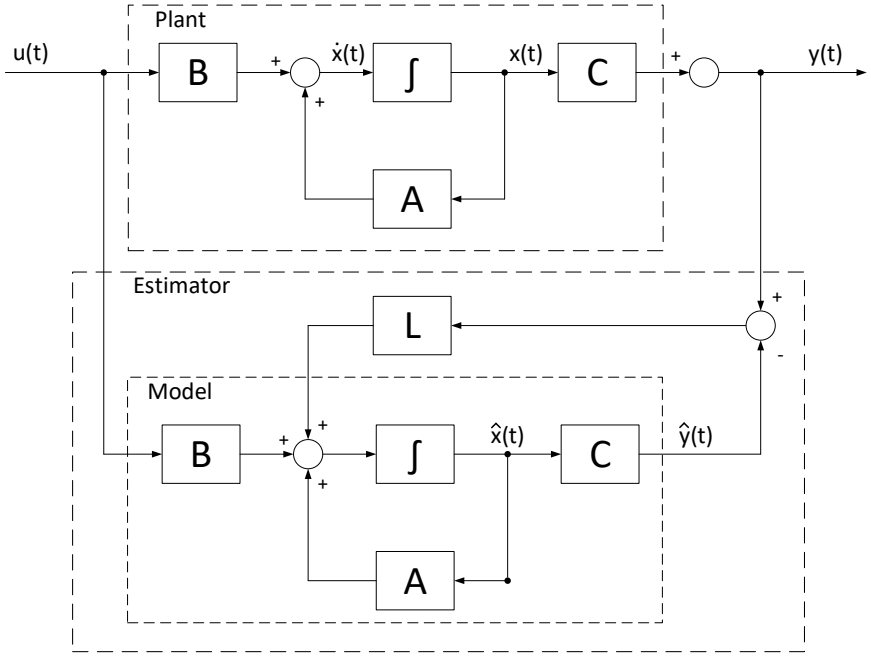


Figure 3.14: Block Diagram of an Estimator Configuration

The dynamics of the error $\dot{\tilde{x}}$ is given by the difference of the actual state dynamics $\dot{x}$ and the estimated dynamics $\dot{\hat{x}}$.

$$\dot{\tilde{x}} = \dot{x} - \dot{\hat{x}} \tag{3.141}$$

Inserting for Eq. 3.91 and Eq. 3.139, gives the following expression for the error equation.

$$\dot{\tilde{x}} = [Ax + Bu] - [A\hat{x} + Bu + L(y - \hat{y})] \tag{3.142}$$

$$= (A - LC)\tilde{x} \tag{3.143}$$

The characteristic equation of the error is given by.

$$det[sI - (A - LC)] = 0 \tag{3.144}$$

The corrector gain L can be selected with standard methods of pole placement, where the poles should be assigned further left on the left-half-plane (LHP) than the dominant poles of the system to ensure fast and stable eigenvalues. This will force the error $\tilde{x}$ to decay towards zero and remain there, i.e., the estimated states $\hat{x}$ will converge towards the actual state x.

3.4.10 Kalman Filter

In this thesis, the Kalman Filter will be utilized as the estimator. The Kalman Filter is a widely used filter which is applicable in many scenarios. Using the filter as a state-estimator is especially favorable for systems exposed to stochastic noise, and opposed to a constant feedback gain L the Kalman Filter utilizes a varying feedback gain and a recursive method of estimating the states.

A system exposed to process noise w and measurement noise v is given by the general state-space system equations.

$$\dot{x} = Ax + Bu + w \tag{3.145}$$
$$y = Cx + v \tag{3.146}$$

Gaussian noise w and v are independent of each other and are distributed by a zero-mean value, and are assumed to have constant covariance matrices Q and R, respectively [26].

$$p(w) \sim N(0, Q) \tag{3.147}$$
$$p(v) \sim N(0, R) \tag{3.148}$$

The Kalman Filter estimator is given by.

$$\dot{\hat{x}} = A\hat{x} + Bu + L(y - \hat{y}) \tag{3.149}$$

Where the related estimation error e and the error dynamics $\dot{e}$ is given by.

$$e = x - \hat{x} \tag{3.150}$$
$$\dot{e} = (A - LC)e + w - Lv \tag{3.151}$$

The feedback (Kalman) gain L are to be chosen so that the filter aims to minimize the mean square error of the state estimation, described by.

$$P = \lim_{t\to\infty} E\{e^2\} = \lim_{t\to\infty} E\{[x - \hat{x}][x - \hat{x}]^\top\} \tag{3.152}$$

The optimal Kalman Gain used to minimize the error of Eq. 3.152 is expressed as.

$$L = PC^\top R^{-1} \tag{3.153}$$

Where P is found from the solution of the matrix-Riccati-equation.

$$AP + PA^\top - PC^\top R^{-1} CP + Q = 0 \tag{3.154}$$

The Kalman Filter principle can be divided into a *predictor* phase (time-update) and a *corrector* phase (measurement-update), these aim to estimate a state x of a discrete-time controlled process, which is governed by linear stochastic differential equation (Eq. 3.155), using a measurement z (Eq. 3.156) [26]. The *a priori* state estimate which is based on the previous knowledge is denoted with an super-script ($\hat{x}^-$), and the *a posteriori* state estimate is given by the measurement. The following discrete equations will use k to denote the current time step, and $k-1$ for the previous step.

The state-vector is given by.

$$x_k = Ax_{k-1} + Bu_{k-1} + w_{k-1} \tag{3.155}$$

where:

x_k	-	Current state (time step k)
x_{k-1}	-	Previous state (time step k)
A	-	Transformation $x_{k-1} \rightarrow x_k$ (system matrix)
B	-	Transformation $u_k \rightarrow x_k$ (input vector)
w_{k-1}	-	Process noise

The measurement is given by.

$$z_k = Hx_k + v_k \tag{3.156}$$

where:

z_k	-	Current measurement (time step k)
x_k	-	Current state (time step k)
H	-	Transformation $u_k \rightarrow z_k$ (output vector)
v_k	-	Measurement noise

Predictor Equations

During the time-update phase, the *a priori* estimate of the state-vector is calculated together with the error covariance matrix P.

$$\hat{x}_k^- = A\hat{x}_{k-1} + Bu_{k-1} \tag{3.157}$$

$$P_k^- = AP_{k-1}A^\top + Q \tag{3.158}$$

Corrector Equations

In the measurement-update phase the Kalman gain, the *a posteriori* state estimate, and error covariance matrix are calculated.

$$L_k = P_k^- H^\top (HP_k^- H^\top + R)^{-1} \tag{3.159}$$

$$\hat{x}_k = \hat{x}_k^- + L_k(z_k - H\hat{x}_k^-) \tag{3.160}$$

$$P_k = (I - L_k H)P_k^- \tag{3.161}$$

After each cycle of predictor and corrector phase, the process is repeated using the previous *a posteriori* estimates to predict the new *a priori* estimates. An overview of the Kalman Filter operation is illustrated in Fig. 3.15.

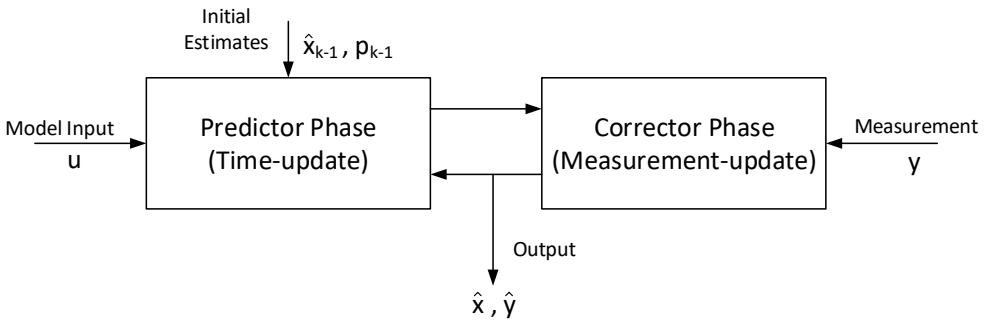


Figure 3.15: Overview of the Kalman Filter Operation

3.4.11 Extended Kalman Filter

As the name suggests, the Extended Kalman Filter (EKF) is an upgrade of the original filter presented in Sec. 3.4.10. Whereas the original Kalman Filter tries to estimate the states of a process governed by linear functions, an EKF can handle non-linear differential equations of the state transition and measurement models.

The EKF exploit methods similar to the characteristics of the Taylor series expansion, where a linearization of the estimation is conducted around the current estimation by utilizing the partial derivatives of the process and measurement functions. Which allows the EKF to compute state-estimation of models with non-linear relationships [26]. The Extended Kalman Filter estimator is based on the following equations for the state-vector and measurement.

$$\dot{\hat{x}} = f(x, u) + w \tag{3.162}$$
$$z = h(x) + v \tag{3.163}$$

Where the process model $f(x, u)$ and measurement model $h(x)$ are non-linear functions of the state-vector and control vector. The process noise w and measurement noise v shares the same assumption made for Eq. 3.147 and Eq. 3.148, where the noise is zero-mean Gaussian noise with covariance Q and R.

The EKF shares the same principle of operating with a predictor phase and update phase, as presented for the standard Kalman Filter, where a superscript $\hat{x}^-$ defines the *a priori* state estimate, and the *a posteriori* state estimate is based on the measurement. The discretized version of the EKF estimator equations Eq. 3.162 and Eq. 3.163, is derived as.

$$\hat{x}_k = f(x_{k-1}, u_{k-1}) + w_k \tag{3.164}$$
$$z_k = h(x_k) + v_k \tag{3.165}$$

Where k and $k-1$ denotes the current and previous time step, respectively. The function of the process model f can be used to calculate the predicted state of the previous estimate, and the measurement function h can be used to compute the predicted measurement from the predicted state. The partial derivative of the functions f and h also referred to as the Jacobian, are used to predict the covariance. The Jacobian matrices defined by Eq. 3.166 and Eq. 3.167 are computed at each time step with the current predicted states, which is essentially the process of linearizing the non-linear function around the current estimate.

$$F_k = \frac{\partial f}{\partial x}\Big|_{\hat{x}_k, u_k} \tag{3.166}$$

$$H_k = \frac{\partial f}{\partial x}\Big|_{\hat{x}_k^-} \tag{3.167}$$

Predictor Equations

The time-update equations compute the *a priori* estimate of the state-vector and the covariance estimate.

$$\hat{x}_k^- = f(\hat{x}_{k-1}, u_{k-1}) \tag{3.168}$$

$$P_k^- = F_k P_{k-1} F_k^\top + Q_k \tag{3.169}$$

Corrector Equations

In the update phase, the *a posteriori* state and covariance estimation are computed with the use of the measurement residual $\tilde{y}_k$, residual covariance S_k and the Kalman gain K_k.

$$\tilde{y}_k = z_k - h(\hat{x}_k^-) \tag{3.170}$$

$$S_k = H_k P_k^- H_k^\top + R_k \tag{3.171}$$

$$K_k = P_k^- H_k^\top S_k^{-1} \tag{3.172}$$

$$\hat{x}_k = \hat{x}_k^- + K_k \tilde{y}_k \tag{3.173}$$

$$P_k = (I - K_k H_k) P_k^- \tag{3.174}$$

4 Method

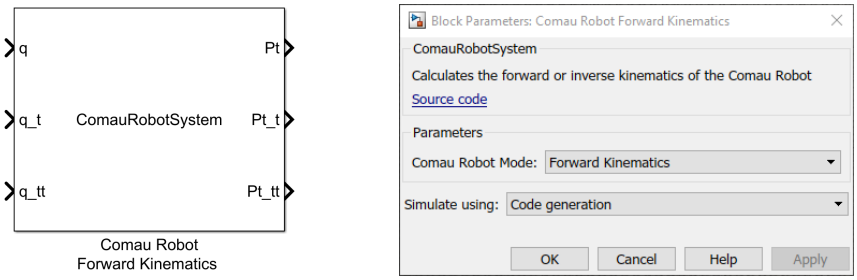
This chapter will introduce the methods used to perform the simulation and control of the anti-swing system. These are based on the fundamental theory previously derived in Ch. 3. Models of the industrial robot, suspended load, and a full system model will be developed. Different controller schemes and their related implementation will be presented. In the end, a simulation model of the anti-swing controller system will be introduced. The system models and controller design are developed with the use of Matlab®and Simulink®.

4.1 Robot Model

Modelling of the industrial robot is divided into two main parts; a system representing the forward kinematic equations (Sec. 3.1.2), and a system for the respective inverse kinematic equations (Sec. 3.1.3). These models are all represented by a Matlab system block, which allows for easy integration to the Simulink environment. As a simplification, the robot is considered to be a fully rigid system. Hence no dynamics will be included in the modelling of the robot system.

4.1.1 Forward Kinematics

The forward kinematic equations for the Comau robot calculates the tool-point motion as a function of the joint angle inputs. A Matlab system block has been developed to compute these kinematic transformations. The block is designed by using the derived equations for Tool-point position (Eq. 3.8), velocity (Eq. 3.11) and acceleration (Eq. 3.14). Figures of the system block in the Simulink environment and the block configuration is shown in Fig 4.1.



(a) System block in the Simulink Environment

(b) System Block Configuration

Figure 4.1: Comau Robot Forward Kinematic System Block

As Fig. 4.1a shows, the system block requires input vectors of the joint angular position q, velocity $\dot{q}$ and acceleration $\ddot{q}$, which corresponds to the expressions of Eq. 3.8, Eq. 3.11 and Eq. 3.14, respectively.

$$q = \begin{bmatrix} q_1 \\ q_2 \\ q_3 \end{bmatrix} = \begin{bmatrix} \theta_1 \\ \theta_2 \\ \theta_3 \end{bmatrix} \tag{4.1}$$

$$\dot{q} = \begin{bmatrix} \dot{q}_1 \\ \dot{q}_2 \\ \dot{q}_3 \end{bmatrix} = \begin{bmatrix} \dot{\theta}_1 \\ \dot{\theta}_2 \\ \dot{\theta}_3 \end{bmatrix} \tag{4.2}$$

$$\ddot{q} = \begin{bmatrix} \ddot{q}_1 \\ \ddot{q}_2 \\ \ddot{q}_3 \end{bmatrix} = \begin{bmatrix} \ddot{\theta}_1 \\ \ddot{\theta}_2 \\ \ddot{\theta}_3 \end{bmatrix} \tag{4.3}$$

In a simulation environment, only one of the input vectors will be assigned with values, and the remaining two will directly be calculated through time differentiation or integration. An example of a simulation configuration is shown in Fig. 4.2, here an additional plotting functionality has been added to visualize the 3D motion response of the model.

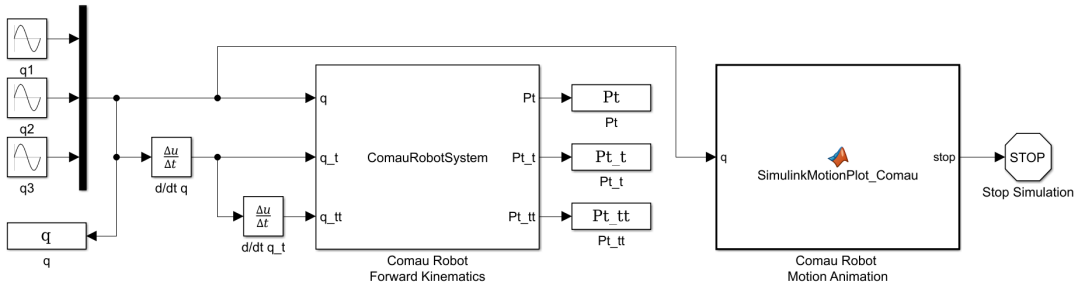


Figure 4.2: Comau Robot Forward Kinematic Model

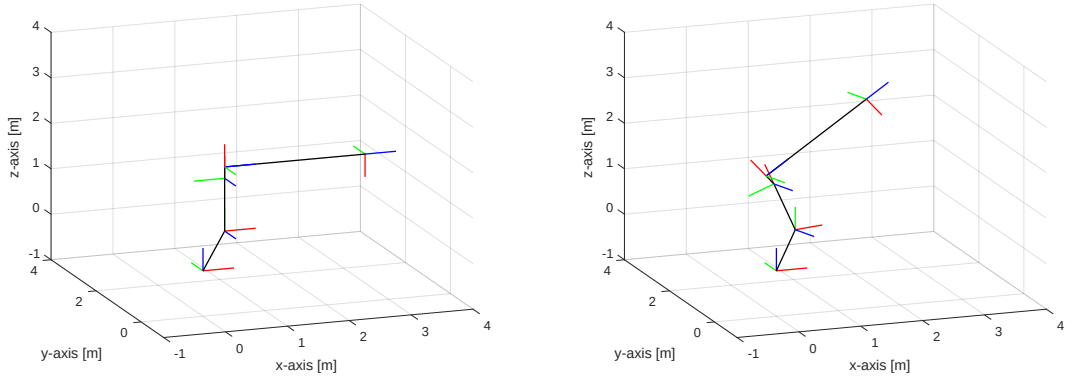
Two simulation examples are conducted on the forward kinematic model, a *home-position* configuration and a custom *offset-position*. Fig. 4.3 shows the 3D visualization of the robot configuration for the two examples. Fig. 4.3a corresponds to the *home-position*, and Fig. 4.3b shows the *offset-position* example. The details of the joint angle inputs and the resulting tool-point position, are described by Tab. 4.1 and Tab. 4.2.

Table 4.1: Robot Forward Kinematic Model - Home Position

Input:			**Output:**		
q	**Value**	**Unit**	P_t	**Value**	**Unit**
θ_1	0	$[deg]$	x_t	2.619	$[m]$
θ_2	0	$[deg]$	y_t	0	$[m]$
θ_3	-90	$[deg]$	z_t	2.240	$[m]$

Table 4.2: Robot Forward Kinematic Model - Offset Position

Input:			**Output:**		
q	**Value**	**Unit**	P_t	**Value**	**Unit**
θ_1	-15	$[deg]$	x_t	1.612	$[m]$
θ_2	-20	$[deg]$	y_t	0.432	$[m]$
θ_3	-55	$[deg]$	z_t	3.426	$[m]$



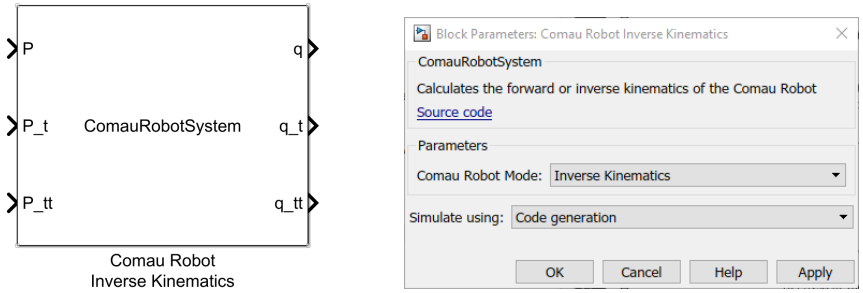
(a) Robot Configuration in Home Position (b) Robot Configuration in Offset Position

Figure 4.3: 3D Visualization from the Comau Robot Forward Kinematic Model

The Matlab scripts for the system block of the forward kinematic model and the 3D motion animation, are available in App. C.1.2 and App. C.7.1, respectively.

4.1.2 Inverse Kinematics

Opposite to the forward kinematics, the inverse kinematic of the Comau robot calculates the joint motion required to realize the desired tool-point motion. The system block representing the inverse kinematics model, utilizes the derived equations from Sec. 3.1.3. The joint angle vector is calculated by using Eq. 3.15, Eq. 3.27, and Eq. 3.29, the angular velocity and acceleration are derived from Eq. 3.30 and Eq. 3.31. Figures of the system block for the inverse kinematics and the parameters configuration are shown in Fig. 4.4.



(a) System block in the Simulink Environment (b) System Block Configuration

Figure 4.4: Comau Robot Inverse Kinematic System Block

In the same manner, as for the forward kinematic system block, only one input vector will be assigned in a simulation environment, where the remaining can be calculated by time differentiation or integration. The Matlab script for the system block is available in App. C.1.2.

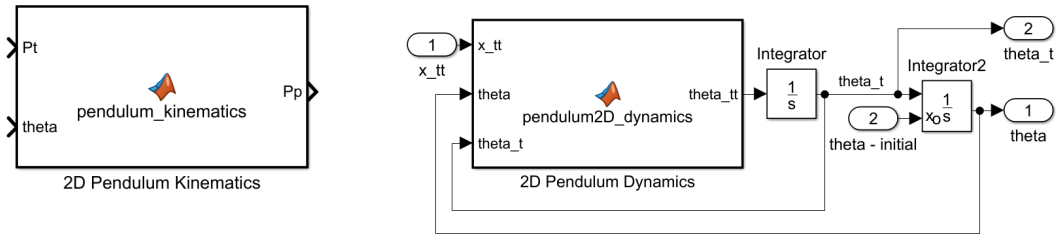
4.2 Suspended Load Model

The system model of the suspended load is based on the equations derived in Sec. 3.2. This section is divided into two parts, where firstly a model of the simple pendulum (2D-system) is made, and secondly, a full model of the 3D system is developed. The latter will be combined with the system model of the Comau robot, where the tool-point is responsible for the actuation of the pendulum.

4.2.1 Simple Pendulum 2D Model

A model of the simple pendulum system is developed for testing different controller schemes and will be used as a basis for the 3-dimensional pendulum. The analysis of the 2D pendulum is presented in Sec. 3.2.1, on which the 2D model will be based on.

The kinematics of the simple pendulum is derived by Eq. 3.35, a Matlab function, seen in Fig. 4.5a, is used to implement this equation into a simulation environment. The equation of motion Eq. 3.45, is a second order differential equation which describes the pendulum dynamics. This system can be solved by numerical integration, which is shown in Fig. 4.5b. The dynamics function requires initial values for the angular position θ_0 and velocity $\dot{\theta}_0$, where the latter is set to zero for all simulation scenarios



(a) Kinematics Function (b) Dynamics Function

Figure 4.5: Functions Describing the Simple Pendulum System

Combining the kinematic and dynamic function gives a system model of the 2D pendulum, this is shown in Fig. 4.6, where an additional plotting function is added to visualize the response of the simulation.

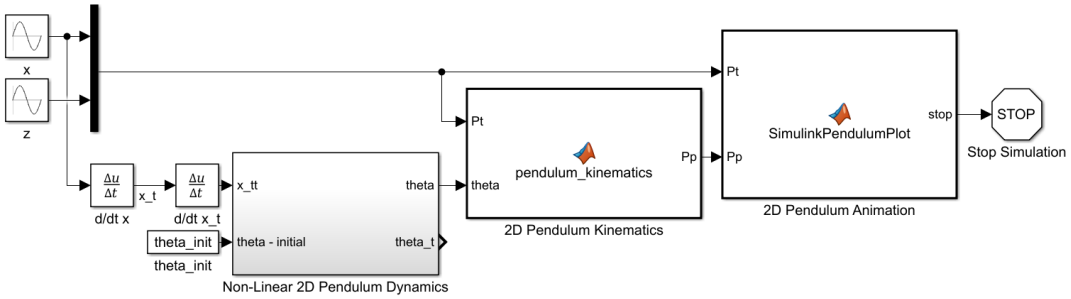


Figure 4.6: Pendulum 2D System Model

A 2D visualization of an example simulation is shown in Fig. 4.7, here the pendulum is initiated with an starting angle of $\theta_0 = 15$ $[deg]$

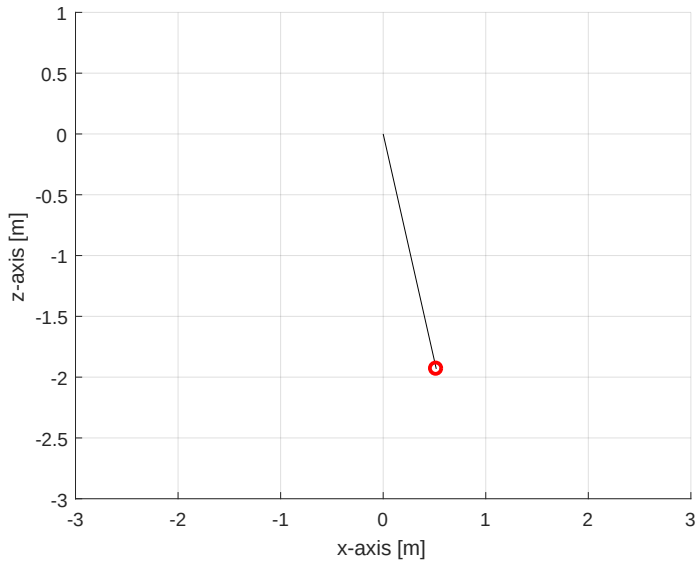


Figure 4.7: Visualization of the 2D Pendulum Model Response

4.2.2 Suspended Load 3D Model

The system model for the 3-dimensional suspended load/pendulum aims to describe the pendulum position as a function of the Euler-angles and tool-point position. The sets of equations describing the 3D system are derived and presented by Sec. 3.2.2, here it can be seen that the pendulum can be described as a coupled system of the dynamics and kinematics.

To implement the system of the 3D pendulum into a simulation environment, the model is constructed as a set of Matlab functions. Allowing for an easy connection with the model describing the Comau robot Sec. 4.1.

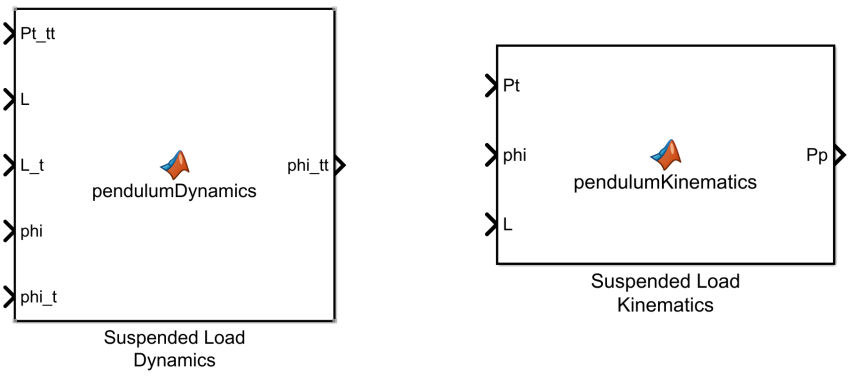
Dynamics

The function describing dynamics of the 3D pendulum is based on the equation of motion given by Eq. 3.56 and Eq. 3.57. This set of equations describes a coupled pair of the second order differential equation for the Euler-angles, which are dependent on the tool-point acceleration $\ddot{P}_t$, and the wire length L_w with its corresponding rate of change $\dot{L}_w$.

A figure of the Matlab functino describing the suspended load dynamics is shown by Fig. 4.8a, the related Matlab script can be found in App. C.2.1.

Kinematics

Suspended load kinematics aims to describe the payload position as a function of the Euler-angles, wire length and tool-point position, which is given by Eq. 3.49. The model of the pendulum kinematics designed as a Matlab function, shown in Fig. 4.8b. The related Matlab script is found in App. C.2.2.



(a) Dynamics Function (b) Kinematics Function

Figure 4.8: Suspended Load/Pendulum System Functions

System Model

The solution of the dynamic function describing the non-linear differential equations can be solved in Simulink by numerical integration (Eq. 3.56 and Eq. 3.57). For this to be computed, a vector of initial positions and velocities for the Euler-angles are required. The vector ϕ_0 containing the initial position angles describes the starting offset of the pendulum, and the initial velocity vector $\dot{\phi}_0$ will be considered to be equal zeros for all simulation scenarios. The wire length's rate of change $\dot{L}_w$ is modelled as a simple time derivation of the wire length L_w.

$$\phi_0 = \begin{bmatrix} \phi_{\alpha,0} \\ \phi_{\beta,0} \end{bmatrix} \tag{4.4}$$

$$\dot{\phi}_0 = \begin{bmatrix} \dot{\phi}_{\alpha,0} \\ \dot{\phi}_{\beta,0} \end{bmatrix} = 0 \tag{4.5}$$

$$\dot{L}_w = \frac{d}{dt}(L_w) \tag{4.6}$$

Combining the dynamic and kinematic functions of the 3D pendulum yields a system model of the suspended load, which is available for use in a simulation. A figure of the 3D suspended load/pendulum system is shown in Fig. 4.9.

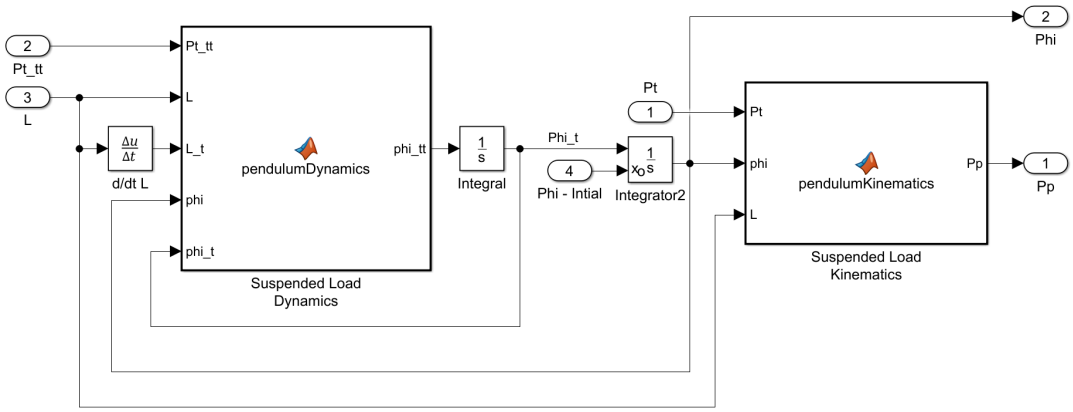


Figure 4.9: Suspended Load/Pendulum 3D System Model

The pendulum position P_p and Euler-angles ϕ are calculated by inputs from the tool-point position P_t and acceleration $\ddot{P}_t$, wire length L_w and the initial conditions of the euler angles ϕ_0.

Suspended Load and Robot System

Combing the model of the 3D pendulum system (Fig. 4.9) with the Comau robot's forward kinematics model (Fig. 4.2) yields a system for the suspended load connected to the tool-point of the robot. A figure of this combined model is shown in Fig. 4.10, an animation plot is added to visualize the motion response in 3D (the Matlab script used to visualize the 3D pendulum is available in App. C.7.2).

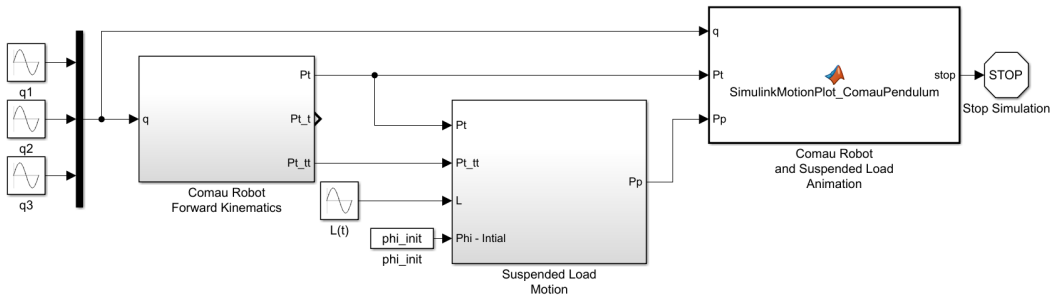


Figure 4.10: Simulation Model of the Robot and Suspended Load

This system model requires inputs from the joint angles q (Eq. 4.1), wire length L_w and the initial conditions for the euler angles ϕ_0 (Eq. 4.4).

A 3D visualization result of the system model in *home-position* is shown in Fig. 4.11, the corresponding input and output values are described in Tab. 4.3.

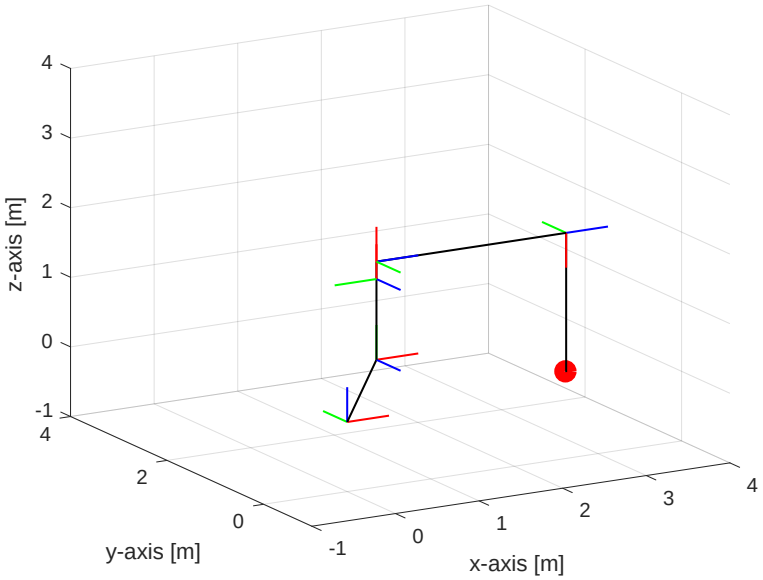


Figure 4.11: 3D Visualization of the Robot and Suspended Load Model in Home Position

Table 4.3: Robot and Suspended Load Model in Home Position

Input:			**Output:**		
Parameter	**Value**	**Unit**	**Parameter**	**Value**	**Unit**
θ	$[0\,,0\,,-90]$	$[deg]$	P_t	$[2.619\,,0\,,2.240]$	$[m]$
L_w	2.0	$[m]$	ϕ	$[0\,,0]$	$[deg]$
ϕ_0	$[0\,,0]$	$[deg]$	P_p	$[2.619\,,0\,,0.240]$	$[m]$

4.3 Motion System Model

With models developed for the robot and the suspended load, the next step is to combine these with the motion of the Stewart platform. The aim is to create a system model which inherits the configuration shown in Fig. 3.8.

4.3.1 Stewart Platform Motion

The system model of the Stewart platform motion is based on the theory and formulations presented by Sec. 3.3.2. The vector η, which contains the parameters for the platform orientation, will is used as the control input for the platform motion.

The relations between input η and the relative velocity and acceleration are computed by the Matlab function for the Stewart platform motion, which is shown in Fig. 4.12.

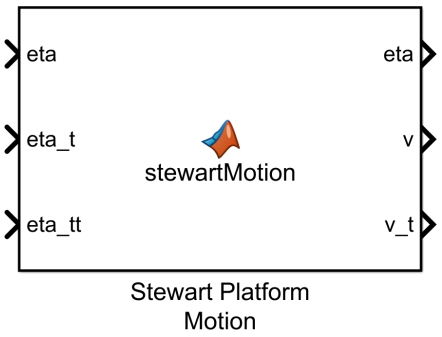


Figure 4.12: Stewart Platform Motion System

This function is based on Eq. 3.69 and Eq. 3.75, where the related Matlab script can be found in App. C.3.

To generate a wave motion for the full system, a sinusoidal signal is used as an input to each element of the orientation vector η. The parameters used in a simulation environment can are listed in Tab. 4.4 for each DOF of the Stewart platform. Eq. 4.7 describes the sinusoidal function used to generate these trajectories, where t denotes the time.

$$y = A\sin(2\pi ft + \sigma\frac{\pi}{180}) \tag{4.7}$$

Table 4.4: Wave Trajectory for the Stewart Platform Motion η

Parameter	**Amplitude** A $[m]$	**Frequency** f $[Hz]$	**Phase** σ $[deg]$
x	0.1	0.1	0
y	0.2	0.1	45
z	0.2	0.1	90
θ	$5\pi/180$	0.1	0
ϕ	$5\pi/180$	0.15	90
ψ	$5\pi/180$	0.2	120

4.3.2 Updated Robot Tool-Point Motion

As described in Sec. 3.3.3, the Stewart platform motion will induce relative motion of the robot base frame, hence a set of transformation equations is introduced to describe the tool-point motion relative to the Stewart platforms neutral frame.

The updated model for the Comau Robot's forward kinematics is shown in Fig. 4.13, where the tool-point position, velocity an acceleration are calculated using Eq. 3.79, Eq. 3.81 and Eq. 3.82, respectively.

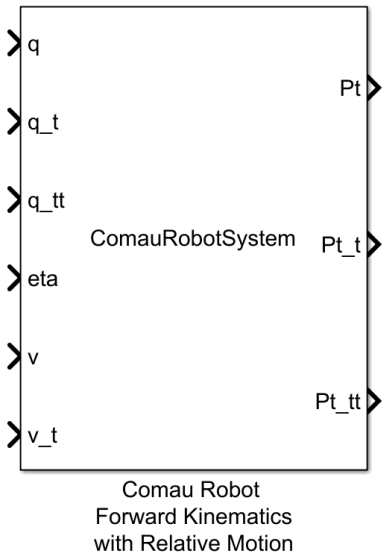


Figure 4.13: Comau Robot Forward Kinematic System, Updated for Relative Motion

As can be seen in the figure, the updated model requires information of the relative motion in addition to the joint inputs. The Matlab script describing the contents of this system is available in App. C.1.2.

Combining the Stewart platform motion model (Fig. 4.12) and the updated system for the forward kinematics of the robot (Fig. 4.13), yields a system model which can be used to compute the tool-point motion relative to the neutral frame of the Stewart platform. A figure of this system is shown in Fig. 4.14.

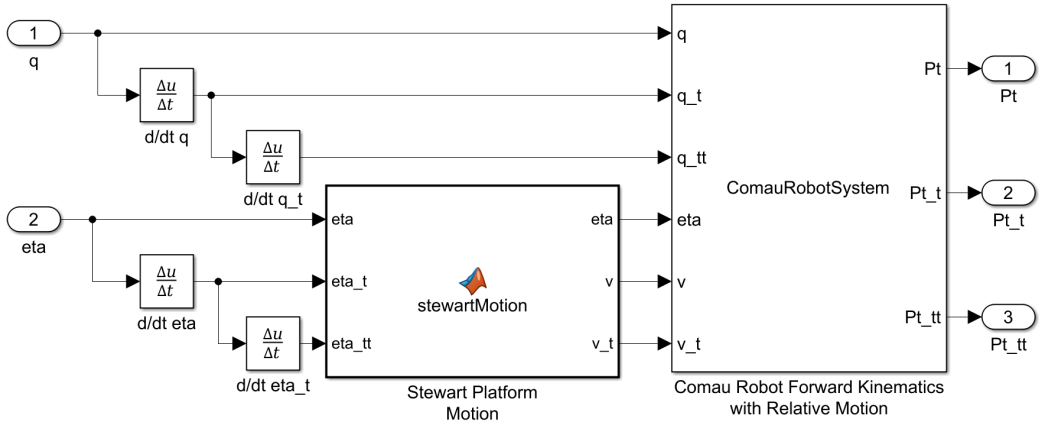


Figure 4.14: Comau Robot Forward Kinematic System with Stewart Platform Motion

4.3.3 Combined Motion System Model

A full 3D system model can now be constructed by combining the Stewart platform and Comau robot motion model (Fig. 4.14) with the suspended load system. The model of the suspended load system will remain equal to the one defined earlier (Fig. 4.9), where the tool-point motion will be given as relative to the neutral frame of the Stewart platform. A figure of the full system model is shown in Fig. 4.15, where the system requires inputs of the joint angle q and the relative platform position η, the Matlab script used to animate the full motion system is given in App. C.7.4.

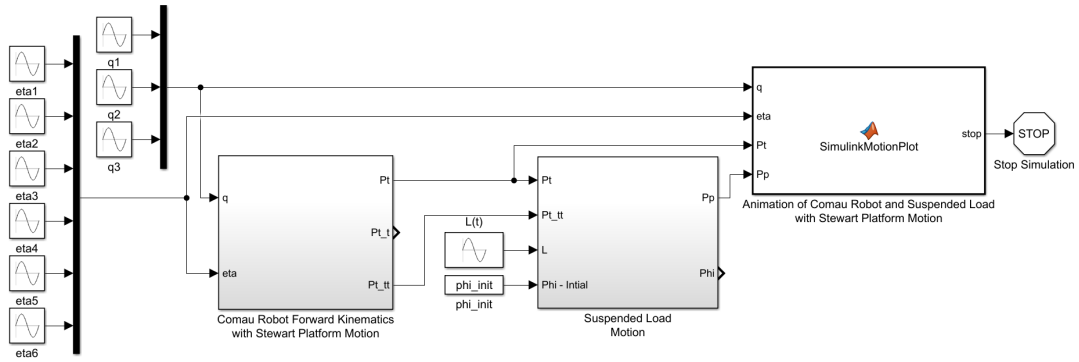


Figure 4.15: Full Motion System Model

A 3D visualization of the full motion system is shown in Fig. 4.16, the robot is in a home configuration, and some small initial values of η are added to visualize the offset for the Stewart platform relative to its the neutral frame. See Fig. 3.8, for comparison and notation of the body frames.

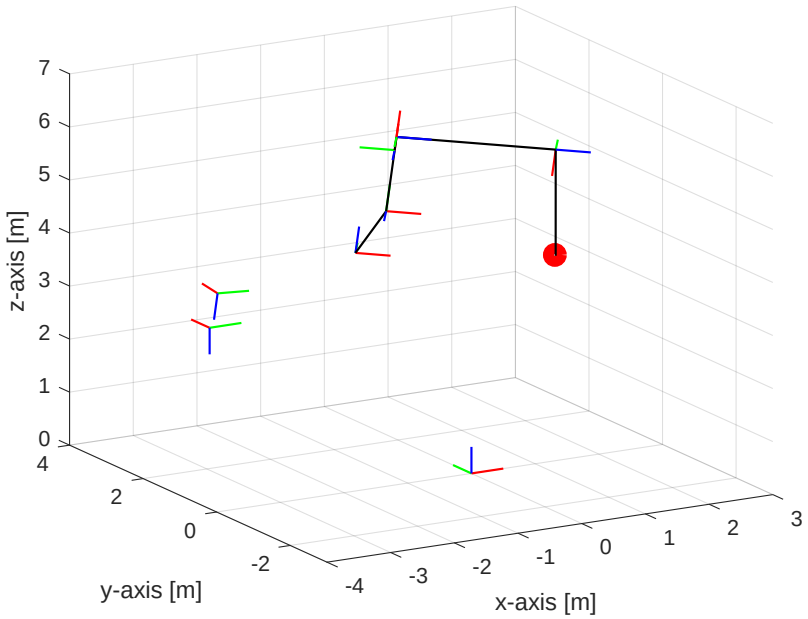


Figure 4.16: 3D Visualization of the Full System Model

4.4 Control System Design Simple Pendulum

The following section will present the methods used to develop different controller schemes for the 2-dimensional pendulum system. It is desired to establish a control system which can actively compensate for the swing angle and the tool-point position.

4.4.1 Non-Linear System Plant

The system plant for the non-linear pendulum system is shown in Fig. 4.17. The plant uses the dynamics function of Fig. 4.5b to describe the equation of motion given by Eq. 3.45. The tool-point horizontal position and velocity is also given by the plant, but for simplicity, no dynamics are added to these terms.

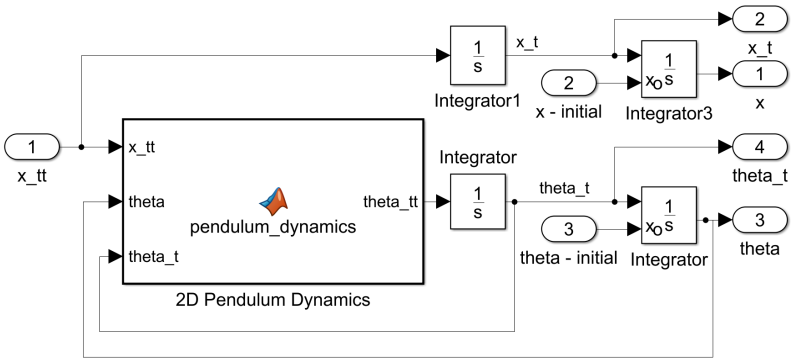


Figure 4.17: Non-Linear Plant Pendulum 2D

The system plant for the 2D pendulum acts as a single-input multi-output (SIMO) system, where the respective state-vector and system input is given by.

$$x = \begin{bmatrix} x_t \\ \dot{x}_t \\ \theta \\ \dot{\theta} \end{bmatrix} \tag{4.8}$$

$$u = \ddot{x}_t \tag{4.9}$$

An assumption is made, where the tool-point horizontal position x_t and the angle θ are considered as measurable states. The describing equations of the non-linear system can be expressed as.

$$\dot{x} = f(x,u) = \begin{bmatrix} \dot{x}_t \\ u \\ \dot{\theta} \\ (-u\cos\theta - g\sin\theta)/L_w \end{bmatrix} \tag{4.10}$$

$$y = h(x) = \begin{bmatrix} x_t \\ \theta \end{bmatrix} \tag{4.11}$$

4.4.2 Linearization

The concept of state-space linearization, as presented by Sec. 3.4.2, is utilized to represent the non-linear plant as state-space system. The equilibrium point, for which the plant is linearized around, is found by setting the input equal to $u_0 = 0$, corresponding to zero acceleration. The equilibrium values for the state vector x can be calculated by setting $\dot{x}_0 = f(x_0, u_0) = 0$.

$$\dot{x}_0 = f(x_0, u_0) = \begin{bmatrix} \dot{x}_t \\ u_0 \\ \dot{\theta} \\ (-u_0 \cos\theta - g \sin\theta)/L_w \end{bmatrix} = 0 \tag{4.12}$$

Which gives the following equilibrium state vector

$$x_0 = \begin{bmatrix} 0 \\ 0 \\ k\pi \\ 0 \end{bmatrix} \quad , \quad k : integer \tag{4.13}$$

The pendulum is in a equilibrium state when it is pointing downwards $\theta = 0$ or pointing upwards $\theta = \pi$, the former value is chosen as an equilibrium state for this system design. The state-space matrices A, B, C, and D, evaluated at x_0 and u_0 can now be calculated by Eq. 3.104 - 3.107.

$$A = \left[\frac{\partial f(x,u)}{\partial x}\right]_{x_0,u_0} = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -g/L_w & 0 \end{bmatrix} \tag{4.14}$$

$$B = \left[\frac{\partial f(x,u)}{\partial u}\right]_{x_0,u_0} = \begin{bmatrix} 0 \\ 1 \\ 0 \\ -1/L_w \end{bmatrix} \tag{4.15}$$

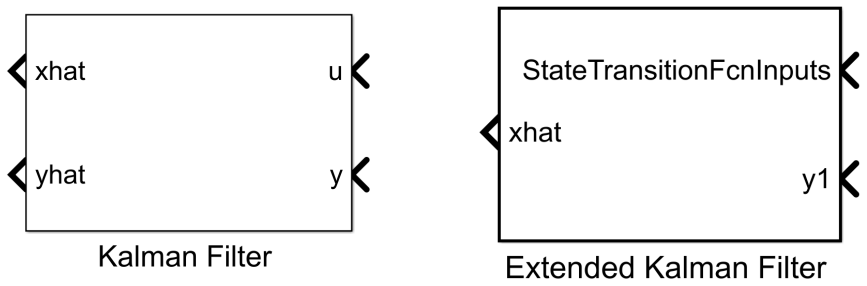
$$C = \left[\frac{\partial h(x,u)}{\partial x}\right]_{x_0,u_0} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 \end{bmatrix} \tag{4.16}$$

$$D = \left[\frac{\partial h(x,u)}{\partial u}\right]_{x_0,u_0} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{4.17}$$

The wire length is considered to be of a constant length $L_w = 2.0$ $[m]$, and gravity equals $g = 9.81$ $[\frac{m}{s^2}]$ for the simulation of the 2-dimensional pendulum.

4.4.3 Estimator Design

In the control design of the simple pendulum, both an original Kalman Filter (KF) and an Extended Kalman Filter (EKF) will be developed to estimate the states of the system. This implementation is conducted due to the interest of comparing the estimation results of the two estimators.



(a) Kalman Filter Block (b) Extended Kalman Filter Block

Figure 4.18: Simulink Representation of the Estimator Blocks

Kalman Filter

The linear Kalman Filter estimator is based on the presented theory in Sec. 3.4.10, where the A and B matrices are found from the state-space linearization. In a simulation environment, the available Kalman Filter block (see Fig. 4.18a) will be used. This block requires inputs for the state-space system (state-space matrices found from the linearization). In addition to a guess on the initial states, information of the covariance matrices for the initial state P, process noise Q and measurement noise R is also required by the Kalman Filter block.

Extended Kalman Filter

The Extended Kalman Filter, described by Sec. 3.4.11, uses the non-linear equation of the plant (Eq. 4.10 and Eq. 4.11) to estimate the system states. In the simulation of the 2D pendulum controller, the Extended Kalman Filter will be implemented as a Simulink block, which is shown in Fig. 4.18b. Similar to the standard Kalman Filter, the extended version requires inputs for the covariance matrices of the initial state P, process noise Q and measurement noise R.

The state-transition function $f(x, u)$ and measurement function $h(x, u)$ are given to the block as Matlab functions. As an optional input, the block can be supplied with the respective Jacobian functions F and H, described by Eq. 3.166 and Eq. 3.167, respectively. But for the relative simple problem of the 2D pendulum, these are chosen to be computed numerically.

Comparison

A simulation model is developed to compare standard Kalman Filter and the Extended version. Values for the covariance matrices are found through experiments and are identical for the two estimators. Numerical values for the process noise covariance Q, measurement noise covariance R, and initial state covariance P is given by.

$$Q = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0.001^2 & 0 & 0 \\ 0 & 0 & 0.01^2 & 0 \\ 0 & 0 & 0 & 0.01^2 \end{bmatrix} \tag{4.18}$$

$$R = \begin{bmatrix} 0.001^2 & 0 \\ 0 & 0.01^2 \end{bmatrix} \tag{4.19}$$

$$P = 1e-3 \tag{4.20}$$

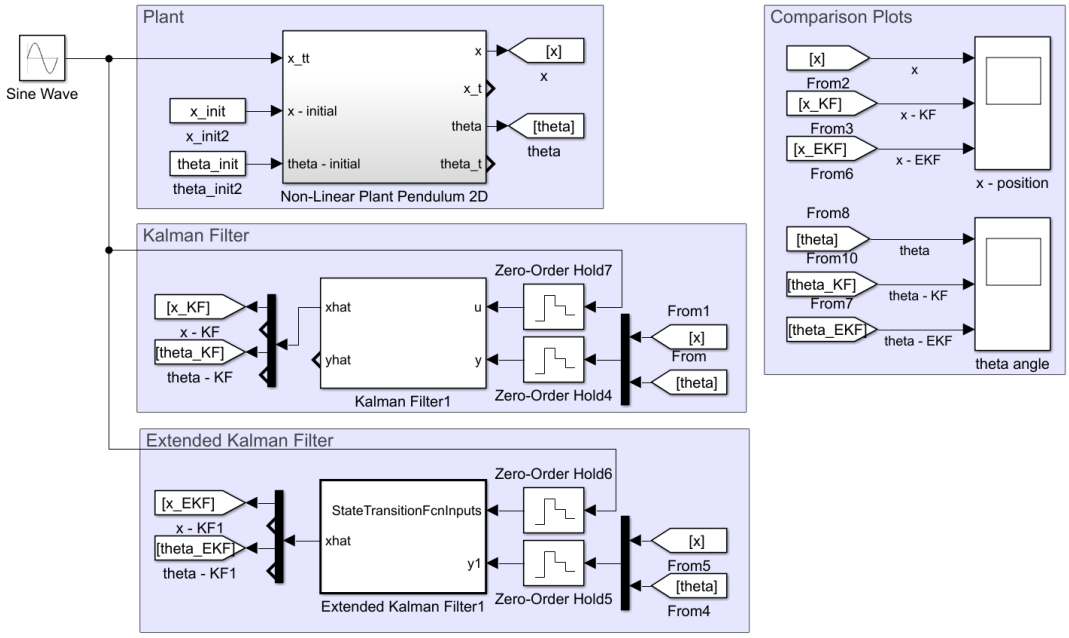


Figure 4.19: 2D Pendulum Model with Kalmen Filter and Extended Kalman Filter

A figure of the simulation model is shown in Fig. 4.19, the non-linear plant of the 2D pendulum is given a sinusoidal acceleration input, and an initial offset angle of $\theta = 20$ $[deg]$. For this scenario, both estimators manage to give an acceptable estimation of the pendulum angle, where the standard Kalman Filter uses a longer time period to converge to the true value. Which is illustrated by the plot of Fig. 4.20.

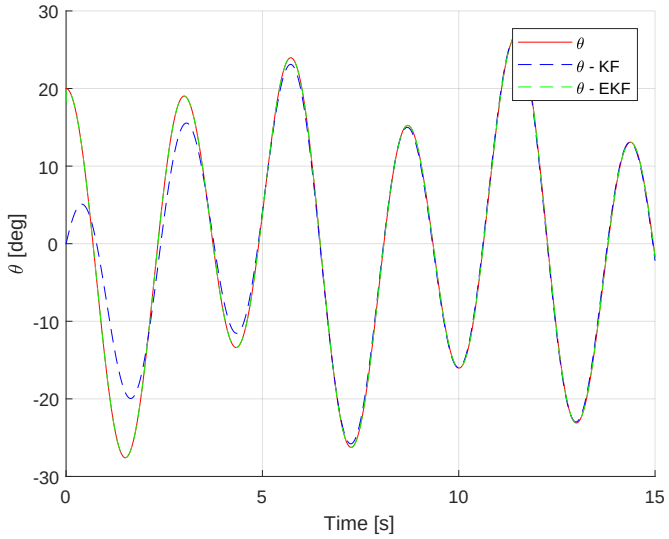


Figure 4.20: Estimator Comparison of 2D Pendulum with Sinusoidal Acceleration Input with $\theta = 20$ $[deg]$

A new scenario is tested, where the input remains the same, but the wire length of the plant is changed to $L_w = 4$. The result are shown in Fig. 4.21, where it can be seen that the standard Kalman Filter no longer give a satisfactory estimation of the pendulum angle.

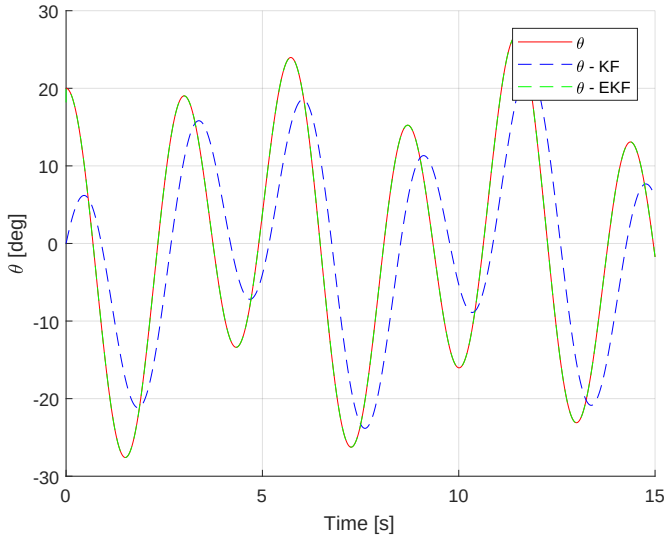


Figure 4.21: Estimator Comparison of 2D Pendulum with Sinusoidal Acceleration Input, $\theta = 20$ $[deg]$ and $L_w = 4$

Based on these results, a decision is made to use the Extended Kalman filter as an estimator for the anti-swing controller.

4.4.4 Linear Control

To compensate for the swing angle of the 2D pendulum, a linear control system is designed based on the state-space linearization derived from Sec. 4.4.2. The linearized model can be described by the general state-space representation.

$$\dot{x} = Ax + Bu \tag{4.21}$$
$$y = Cx + Du \tag{4.22}$$

Where the system matrices A, B, C, and D are defined by Eq. 4.14 - 4.17

Two different controller schemes will be developed for the simple pendulum system, where the aim is to compare their response and behavior.

State-Feedback with Pre-Filter

Firstly a state-feedback controller with a pre-filter for reference input is constructed. The feedback gain K is calculated by using the method of LQR described in detail by Sec. 3.4.5. Values of the weight matrix for the system states Q_c is given by Eq. 4.23. These values are found through experiments where an extra focus is made on penalizing the state of θ. The weight matrix for the control effort is set to $R_c = 1$

$$Q_c = \begin{bmatrix} 10^3 & 0 & 0 & 0 \\ 0 & 10^3 & 0 & 0 \\ 0 & 0 & 10^5 & 0 \\ 0 & 0 & 0 & 10^2 \end{bmatrix} \tag{4.23}$$

The resulting feedback gain equals.

$$K = \begin{bmatrix} 10.00 & 18.10 & -101.88 & 4.78 \end{bmatrix} \tag{4.24}$$

To enable reference tracking a Pre-Filter is added to the controller, where a detailed formulation is given by Sec. 3.4.6. It should be noted that only the state of x_t should be allowed to follow a reference input, hence only the first row of Eq. 4.16 is used in the computation of the pre-filter gain $\bar{N}$ calculated by Eq. 3.122. The system pre-filter gain equals to.

$$\bar{N} = 10 \tag{4.25}$$

The state-feedback system with the added pre-filter is given by the following describing equations.

$$\dot{x} = (\bar{N}r - Kx)B + Ax \tag{4.26}$$
$$y = Cx \tag{4.27}$$

The related closed loop step response is shown in Fig. 4.22.

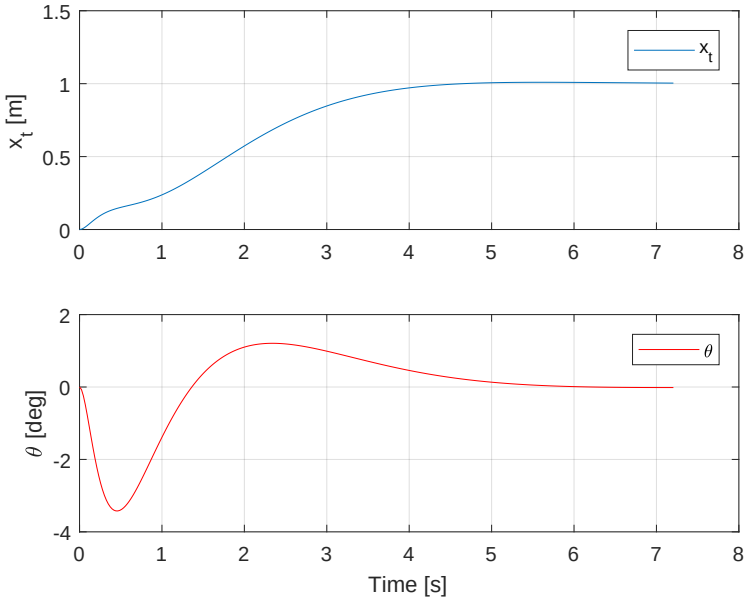


Figure 4.22: Step Response of 2D Pendulum Linear System with State-Feedback and Pre-Filter

The developed state-feedback pre-filter controller can now be combined with the non-linear system plant (Fig. 4.17) and the previously derived Extended Kalman Filter estimator (Fig. 4.18b). The Simulink model of this system is shown in Fig. 4.23.

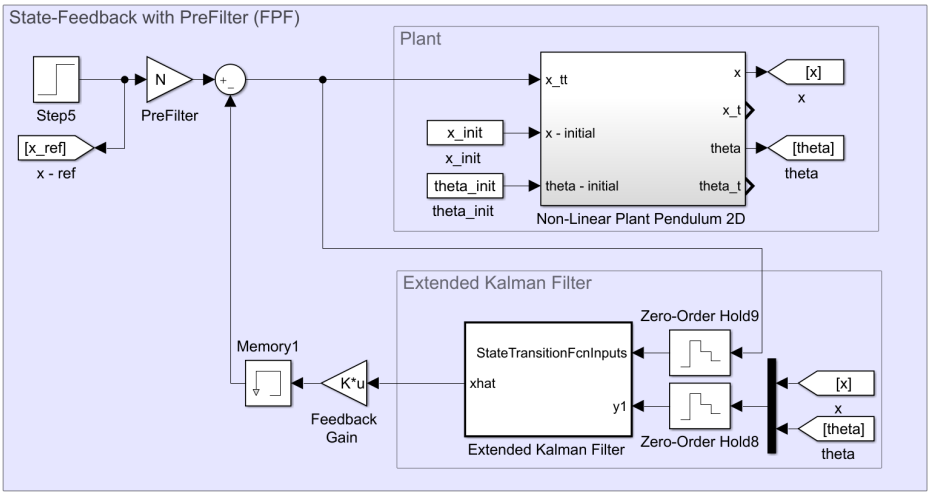


Figure 4.23: Non-Linear 2D Pendulum System with Extended Kalman Filter and State-Feedback Pre-Filter Control

Integral Control

Secondly, an integral control system is designed for the 2D pendulum system. This type of controller adds additional error-states to the system, which in theory will improve the reference tracking compared to pre-filter control. A detailed description of the integral controller is given in Sec. 3.4.7. Introducing the error-states, a new state-space representation of the linearized system is derived.

$$\begin{bmatrix} \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & -C \\ 0 & A \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ B \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{4.28}$$

$$y = \begin{bmatrix} 0 & C \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \tag{4.29}$$

The state-space system description of Eq. 4.28 can be formulated as.

$$\dot{x_i} = A_i x_i + B_i u \tag{4.30}$$

where

$$x_i = \begin{bmatrix} z \\ x \end{bmatrix} \tag{4.31}$$

$$A_i = \begin{bmatrix} 0 & -C \\ 0 & A \end{bmatrix} \tag{4.32}$$

$$B_i = \begin{bmatrix} 0 \\ B \end{bmatrix} \tag{4.33}$$

The original system matrices A, B, C, and D are derived by Eq. 4.14 - 4.17. The feedback law is given by Eq. 3.130, and the feedback gain $K = [-K_e \quad K_o]$ is computed by the LQR method (Sec. 3.4.5), where the new system matrices A_i and B_i, is used. Due to the added error state, an update of the weight matrix for the system state Q_i is required, which is assigned the following values.

$$Q_i = \begin{bmatrix} 10^3 & 0 & 0 & 0 & 0 \\ 0 & 10^3 & 0 & 0 & 0 \\ 0 & 0 & 10^3 & 0 & 0 \\ 0 & 0 & 0 & 10^5 & 0 \\ 0 & 0 & 0 & 0 & 10^2 \end{bmatrix} \tag{4.34}$$

The weight matrix for the control effort is assigned as $R_i = 1$. The resulting feedback gain computed by the LQR method yields the following values.

$$K = [-K_e \quad K_o] \tag{4.35}$$

$$= \begin{bmatrix} 31.62 & 72.38 & 67.02 & -249.17 & 88.49 \end{bmatrix} \tag{4.36}$$

The related step response of the closed loop system is given by Fig. 4.24.

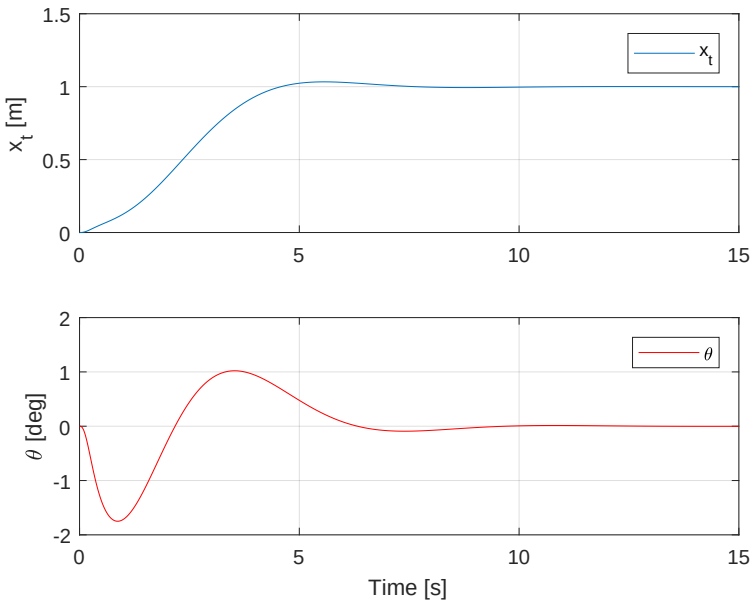


Figure 4.24: Step Response of 2D Pendulum Linear System with State-Feedback and Integral Control

Combining the developed state-feedback integral control with the non-linear plant and using the Extended Kalman Filter as an estimator, yields an anti-swing control system for the 2D pendulum model. The related Simulink model is shown in Fig. 4.25.

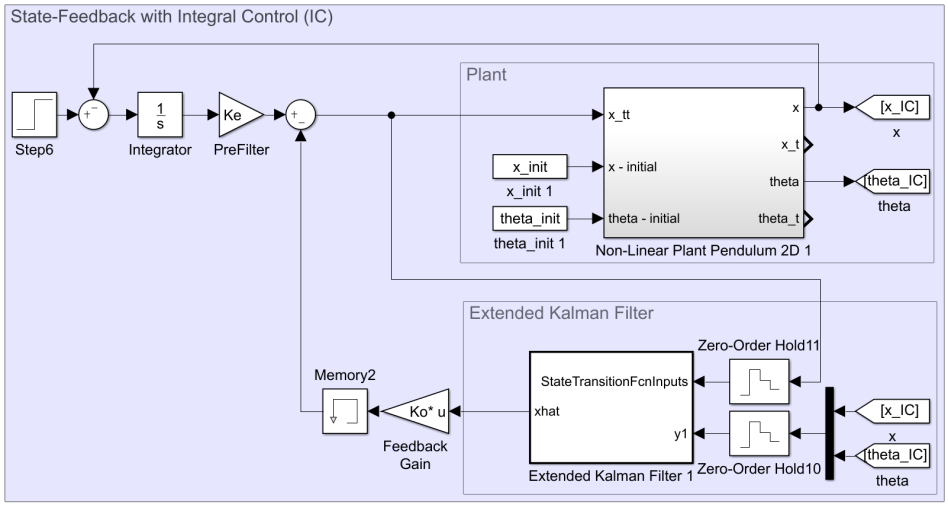


Figure 4.25: Non-Linear 2D Pendulum System with Extended Kalman Filter and State-Feedback Integral Control

The Matlab scripts used to develop the linear controller for the 2D pendulum together with the state-transition and measurement functions for the Extended Kalman Filter are available in App. C.4.

4.4.5 Non-Linear Control

A non-linear control system is developed for the 2D pendulum system. This control design introduces the concept of virtual damping to compensate for swing angle. Here a damping effect is added to the undamped system plant, by manipulation of the input signal.

The differential equation describing the simple pendulum dynamics is given by Eq. 3.45, and can be rewritten as.

$$\ddot{\theta} + \frac{g \sin\theta}{L_w} = -\frac{\ddot{x}_t \cos\theta}{L_w} \tag{4.37}$$

It is desired to alter these dynamics to yield a second order differential equation on the following form.

$$\ddot{\theta} + d\dot{\theta} + k\theta = 0 \tag{4.38}$$

Where d is the damping coefficient. By analyzing Eq. 4.37, it can be seen that the tool-point acceleration $\ddot{x}_t$ is free to use as a virtual damping effect on the original system. Hence, the following formulation is derived to express the desired acceleration needed to dampen out the oscillations.

$$d\dot{\theta} = -\frac{\ddot{x}_t \cos\theta}{L_w} \tag{4.39}$$

$$\Rightarrow \ddot{x}_t = -\frac{L_w d\dot{\theta}}{\cos\theta} \tag{4.40}$$

To implement this virtual damper to the non-linear system plant, the extended Kalman Filter is used to estimate the states of θ and $\dot{\theta}$. A PD (Proportional-Derivative) controller is added for position control, and the Simulink model of the full system is shown in Fig. 4.26.

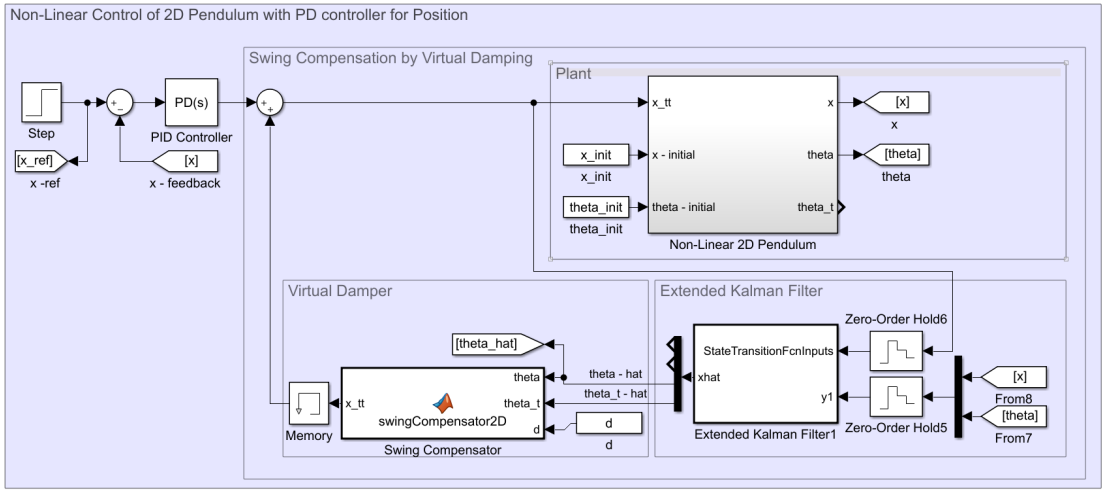


Figure 4.26: Non-Linear 2D Pendulum System with a Non-Linear Virtual Damper Compensator and PD Position Control

A damping coefficient of $d = 5$ is selected for empirical tests, where the values assigned to the PD controller are found by an experimental approach using the Matlab's System Identification and PID tuner toolbox. The corresponding proportional gain and derivative gain is selected as $K_p = 0.2$ and $K_d = 0.85$, which yields the following closed-loop step response.

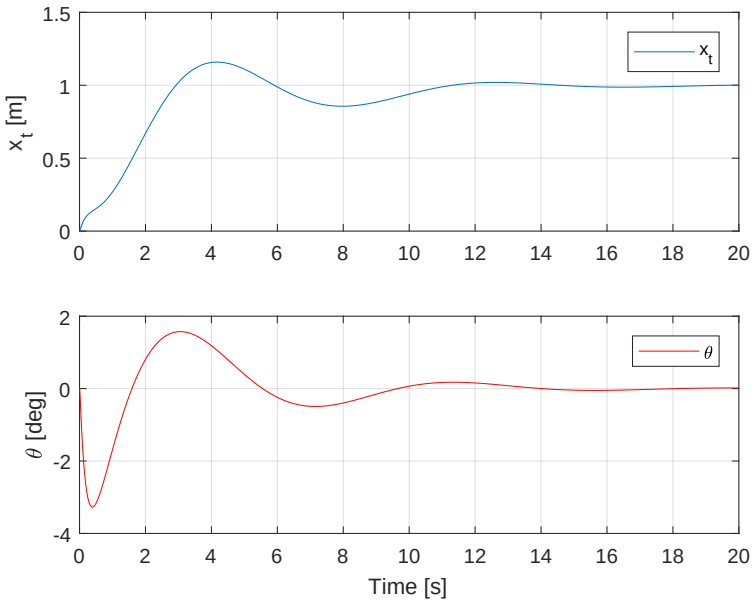


Figure 4.27: Step Response of 2D Pendulum with Non-Linear Virtual Damper and PD Position Control

4.5 Control System Design Suspended Load

This section will present the methods used to develop the different anti-swing controllers for the system consisting of the Comau robot with the attached, suspended load. These controllers will utilize the previously derived simulation models of the robot and suspended load present in Sec. 4.1 and Sec 4.2.2.

The motion of the 3-dimensional pendulum is directly influenced by the motion of the robot's tool-point. Hence it is desired to design a system which is capable of actuating the tool-point. Related to the upcoming control designs, a few assumptions and considerations are made for the suspended load and robot system.

- The Comau robot is considered as a rigid system, where no dynamics are included for the joint actuation.
- Feedback of the tool-point position, and measurements of the suspended load's Euler-angles are assumed to be available.
- The control system for the wire length is not considered in this thesis, and the wire length is assumed to be a constant value of $L_w = 2.0$ $[m]$.

4.5.1 Non-Linear System Plant

Analyzing the equations of motion, given by Eq. 3.56 and Eq. 3.57, it can be seen that the tool-point acceleration components act as the governing parameters to the dynamics of the 3-dimensional suspended load/pendulum. Hence, a control system should be designed such that the tool-point acceleration will operate as the system input. Since the robot is assumed to act as a rigid system, no dynamics will influence the motion of the robot. This is obliviously not the case for the physical system but is made as a simplification.

The Simulink model of the non-linear system is shown in Fig. 4.28, where the suspended load dynamics corresponds to previously defined function, described by Sec. 4.2.2. The system plant requires values for the initial pendulum angles and the initial tool-point position, and can compute the tool-point position and Euler-angles by input from to the tool-point acceleration and wire length.

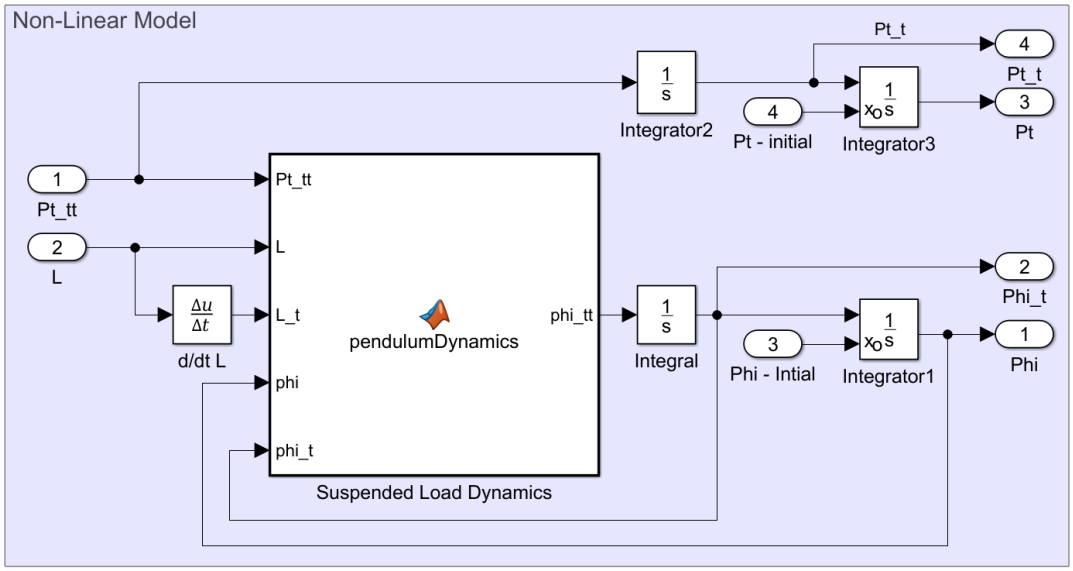


Figure 4.28: Non-Linear Plant of the Suspended Load and Robot Tool-Point

To represent the Comau robot with a full system model, it is possible to implement the sequence of inverse kinematics (Fig. 4.4a) followed by the forward kinematics block (Fig. 4.1a), but since this is just a direct connection, it was omitted for simplification.

The non-linear system describing the pendulum dynamics and the tool-point motion acts as a multi-input multi-output (MIMO) system, where the related state-vector is given by.

$$x = \begin{bmatrix} P_t \\ \dot{P}_t \\ \phi \\ \dot{\phi} \end{bmatrix} \tag{4.41}$$

and the input vector is defined as.

$$u = \ddot{P}_t \tag{4.42}$$

Where:

x	-	State vector	$x(t) \in \mathbb{R}^{10}$
u	-	Input vector	$u(t) \in \mathbb{R}^3$
P_t	-	Tool-point position	$P_t(t) \in \mathbb{R}^3$
$\dot{P}_t$	-	Tool-point velocity	$\dot{P}_t(t) \in \mathbb{R}^3$
$\ddot{P}_t$	-	Tool-point acceleration	$\ddot{P}_t(t) \in \mathbb{R}^3$
ϕ	-	Pendulum Euler-angles	$\phi(t) \in \mathbb{R}^2$
$\dot{\phi}$	-	Pendulum Euler-angular velocity	$\dot{\phi} \in \mathbb{R}^2$

As described by the assumptions, the tool-point position and of the Euler-angles are considered to be measurable states. The describing equations for the non-linear system can be derived as follows.

$$\dot{x} = f(x,u) = \begin{bmatrix} \dot{P}_t \\ u \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} \tag{4.43}$$

$$y = h(x) = \begin{bmatrix} P_t \\ \phi \end{bmatrix} \tag{4.44}$$

Where $\ddot{\phi} = [\ddot{\phi}_\alpha\,,\, \ddot{\phi}_\beta]$ is derived from the equations of motion (Eq. 3.56 and Eq. 3.57).

4.5.2 Extended Kalman Filter Estimator

An Extended Kalman Filter is implemented to estimate the states of the non-linear system. The non-linear estimator is based on the theory presented by Sec. 3.4.11, and uses the derived non-linear functions for the state-transition (Eq. 4.43) and measurement function (Eq. 4.44) to estimate the system states. The estimator is added to the system as a Simulink block (see Fig. 4.18b). For the 3D pendulum system, the respective Jacobian of the state-transition and measurement functions are also supplied to the estimator, the derivation of these Jacobian functions are based on Eq. 3.166 and Eq. 3.167.

Values of the covariance matrices for the initial state P, process noise Q and measurement noise R, are found through an experimental approach to give a satisfactory estimate of the system state. Where the numerical values are given by.

$$P = 10^{-3} \tag{4.45}$$

$$Q = \begin{bmatrix} Q_{P_t} & 0 & 0 & 0 \\ 0 & Q_{\dot{P}_t} & 0 & 0 \\ 0 & 0 & Q_\phi & 0 \\ 0 & 0 & 0 & Q_{\dot{\phi}} \end{bmatrix} \tag{4.46}$$

$$R = \begin{bmatrix} R_{P_t} & 0 \\ 0 & R_\phi \end{bmatrix} \tag{4.47}$$

where the elements of the process noise covariance matrix Eq. 4.46, are given by the following diagonal matrices.

$$
\begin{aligned}
Q_{P_t} &= diag(\,[0.00001^2 \quad 0.00001^2 \quad 0.00001^2]\,) \\
Q_{\dot{P}_t} &= diag(\,[0.001^2 \quad 0.001^2 \quad 0.001^2]\,) \\
Q_{\phi} &= diag(\,[0.001^2 \quad 0.001^2]\,) \\
Q_{\dot{\phi}} &= diag(\,[0.01^2 \quad 0.01^2]\,)
\end{aligned} \tag{4.48}
$$

and the elements of the measurement noise covariance matrix Eq. 4.47, are given by the following diagonal matrix.

$$
\begin{aligned}
R_{P_t} &= diag(\,[0.001^2 \quad 0.001^2 \quad 0.001^2]\,) \\
R_{\phi} &= diag(\,[0.01^2 \quad 0.01^2 \quad 0.01^2]\,)
\end{aligned} \tag{4.49}
$$

Fig 4.29 visualizes the estimator performance of the Euler-angles. Here, a set of different sinusoidal waves has been used as a input to the non-linear dynamic system where the initial angles equal zero ($\phi_\alpha = 0$ and $\phi_\beta = 0$). As the figures shows, the Extended Kalman Filter manages to estimate the angles with a satisfactory result.

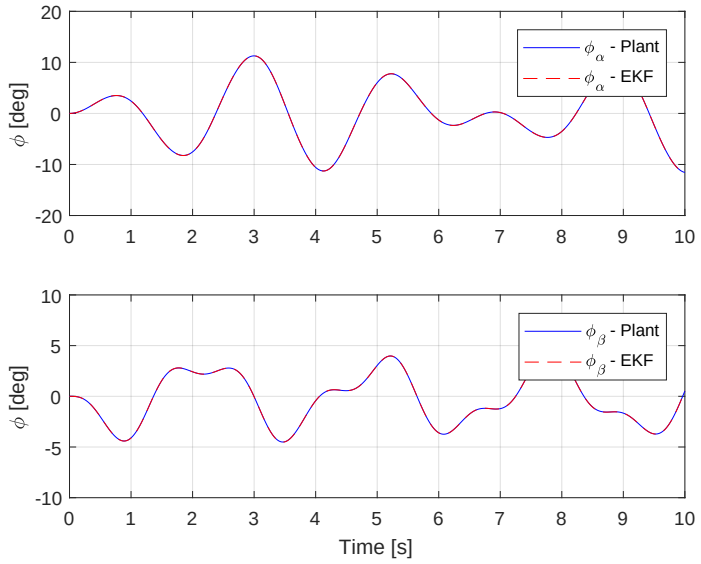


Figure 4.29: EKF Estimation of the Suspended Load System

A second test of the estimator is simulated with the same sinusoidal input. Now the suspended load is initiated with a set of initial offset values equal to $\phi_\alpha = 15$ and $\phi_\beta = 20$. The result is shown in Fig. 4.30. Even though the initial estimator guess is wrong, the Extended Kalman Filter manages to converge to the true value.

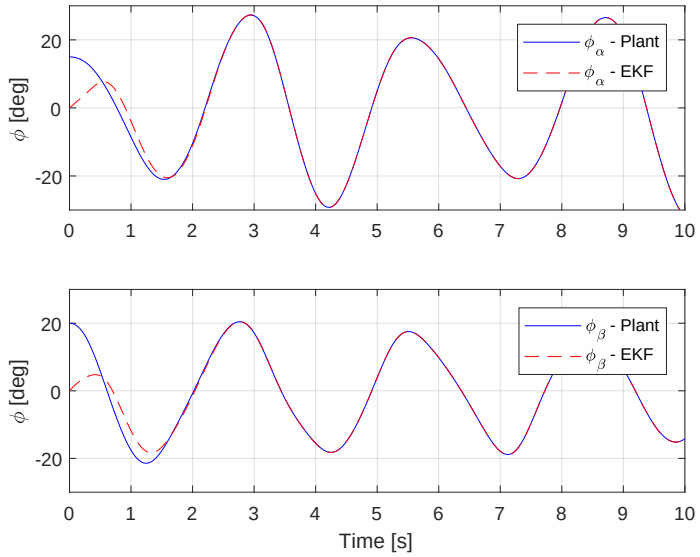


Figure 4.30: EKF Estimation of the Suspended Load System, with Initial Offset Angles

4.5.3 Linearization

It is desired to represent the non-linear plant, described by Eq. 4.43 and Eq. 4.44, as a state-space system. The state-space linearization, based on the presented theory of Sec. 3.4.2, approximates a linear model of the non-linear plant around an equilibrium point. A input of zero acceleration $u_0 = 0$ is chosen, and the following equilibrium values for the system states are found by $\dot{x}_0 = f(x_0, u_0) = 0$, which is derived as.

$$\dot{x}_0 = f(x_0, u_0) = \begin{bmatrix} \dot{P}_t \\ u_0 \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} = 0 \tag{4.50}$$

which yields the following equilibrium state vector.

$$x_0 = \begin{bmatrix} P_t \\ \mathbf{0} \\ \mathbf{0} \\ \mathbf{0} \end{bmatrix} \tag{4.51}$$

Initial angles for the pendulum is assigned to $\phi_\alpha = 0$ and $\phi_\beta = 0$, which corresponds to the pendulum in a downwards position. As can be seen from Eq. 4.51, the initial tool-point position is free, i.e., any reasonable values within the reach of the robot are available for choosing. The state-space system matrices, A, B, C, and D evaluated at x_0 and u_0 can now be computed according to Eq. 4.14 - 4.17.

4.5.4 Linear Control

Compensation for the swing-angles is performed by utilizing the linear system model of the suspended load and robot system. In addition to keep the Euler-angles ϕ_α and ϕ_β as small as possible, the controller should be designed such that the robot tool-point is able to track a reference input. Two different linear control designs will be developed, where it is of interest to compare their responses.

With the state-space linearization, the approximated system model can now be described by the general state space representation, as defined by Eq. 3.91 and Eq. 3.92.

$$\dot{x} = Ax + Bu \tag{4.52}$$
$$y = Cx + Du \tag{4.53}$$

where

$$x = \begin{bmatrix} P_t \\ \dot{P}_t \\ \phi \\ \dot{\phi} \end{bmatrix} \tag{4.54}$$
$$u = \ddot{P}_t \tag{4.55}$$

State-Feedback with Pre-Filter

A state-feedback controller with a pre-filter is the first linear controller to be implemented to the 3D-pendulum and robot system. The feedback gain K is computed by using the method of LQR, described in more detail by Sec. 3.4.5. The weight matrices for the system states Q_c and control effort R_c are designed by an experimental approach until a satisfactory system is met. Extra focus is made on penalizing the state of ϕ, which will force the controller to focus on minimizing the swing angle of the pendulum, the related numerical values for Q_c and R_c are given as.

$$Q_c = \begin{bmatrix} Q_{c,P_t} & 0 & 0 & 0 \\ 0 & Q_{c,\dot{P}_t} & 0 & 0 \\ 0 & 0 & Q_{c,\phi} & 0 \\ 0 & 0 & 0 & Q_{c,\dot{\phi}} \end{bmatrix} \tag{4.56}$$
$$R_c = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{4.57}$$

where the elements of the weight matrix Q_c are given by the following diagonal matrices.

$$\begin{aligned} Q_{c,P_t} &= diag(\begin{bmatrix} 10^3 & 10^3 & 10^3 \end{bmatrix}) \\ Q_{c,\dot{P}_t} &= diag(\begin{bmatrix} 10^2 & 10^2 & 10^2 \end{bmatrix}) \\ Q_{c,\phi} &= diag(\begin{bmatrix} 10^4 & 10^4 \end{bmatrix}) \\ Q_{c,\dot{\phi}} &= diag(\begin{bmatrix} 10^3 & 10^3 \end{bmatrix}) \end{aligned} \tag{4.58}$$

The resulting feedback gain, computed by the LQR method equals.

$$K = \begin{bmatrix} 31.62 & 0 & 0 & 41.07 & 0 & 0 & 236.32 & 0 & 25.55 & 0 \\ 0 & 10.0 & 0 & 0 & 34.28 & 0 & 0 & -76.10 & 0 & -4.82 \\ 0 & 0 & 31.62 & 0 & 0 & 12.78 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{4.59}$$

The added feedback gain K will force the states to a zero steady-state, which is desired for the states related to the pendulum angles. However, to enable reference tracking for the tool-point, a pre-filter is added accordingly to the formulation described in Sec. 3.4.6. As Eq. 3.122 describes, the gain $\bar{N}$ is computed using the output vector C. Since it is desired to only have the states of P_t influenced by a pre-filter gain, only the first three rows of C will be used to compute the gain $\bar{N}$. The resulting numerical values of the pre-filter gain is given by.

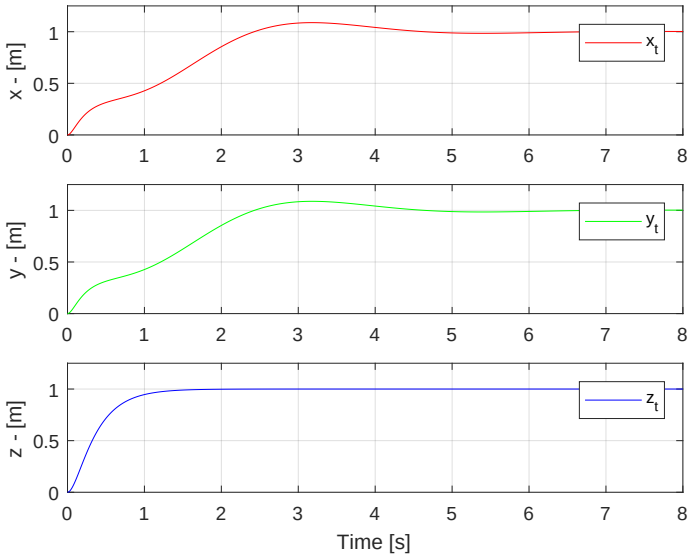
$$N = \begin{bmatrix} 31.62 & 0 & 0 \\ 0 & 10.0 & 0 \\ 0 & 0 & 31.62 \end{bmatrix} \tag{4.60}$$

The system equations describing the state-feedback system with pre-filter are given by.

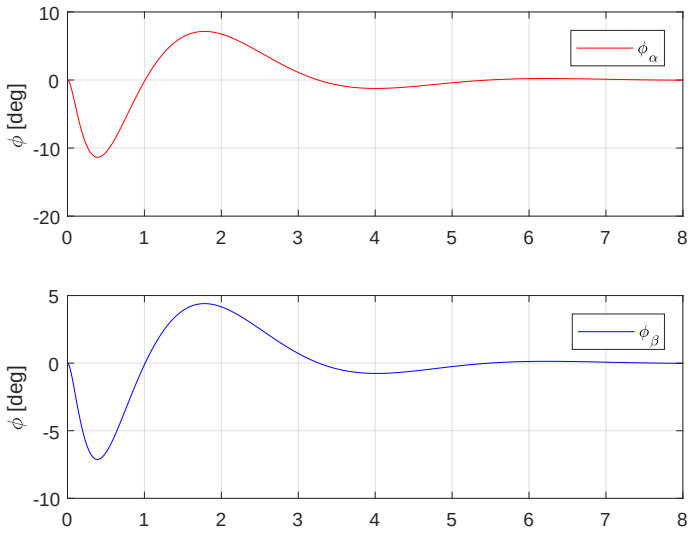
$$\dot{x} = (\bar{N}r - Kx)B + Ax \tag{4.61}$$

$$y = Cx \tag{4.62}$$

The closed loop step response for the state-feedback pre-filter controller, is given for the tool-point position by Fig. 4.31a and the related pendulum angle response can be seen in Fig. 4.31b.



(a) Tool-Point Position Response



(b) Pendulum Angle Response

Figure 4.31: Closed Loop Step Response of the Non-Linear Suspended Load and Robot System, with State-Feedback Pre-Filter Control

The Simulink model of the non-linear suspended load and robot system, with a state-feedback pre-filter control and Extended Kalman Filter estimator, can be seen in Fig. 4.32.

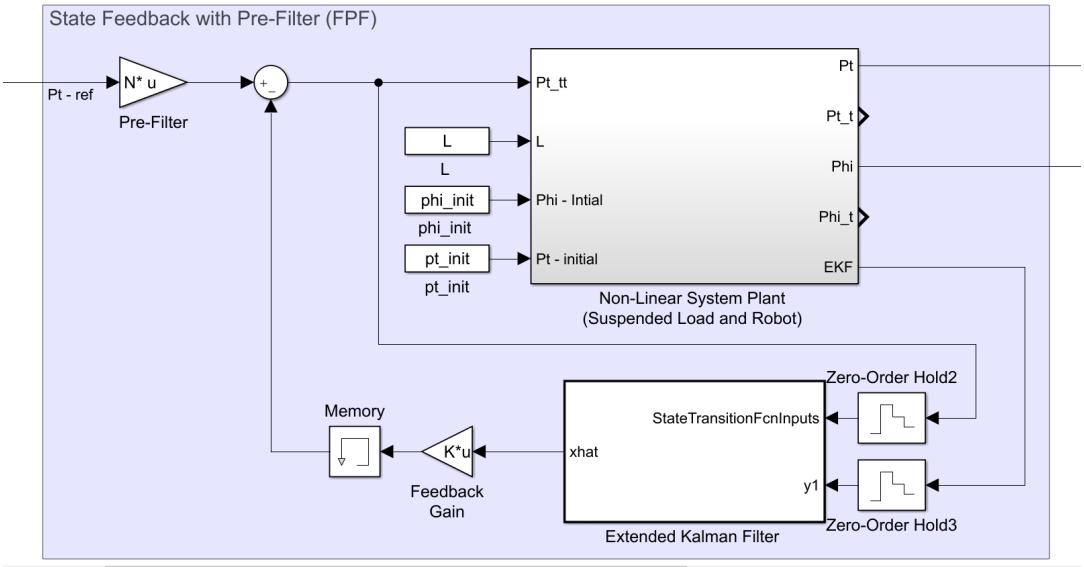


Figure 4.32: Simulink Model of the Non-Linear Suspended Load and Robot System, with State-Feedback Pre-Filter Control

Integral Control

In addition to the developed state-feedback with pre-filter control, it is desired to design an integral controller for the 3D-pendulum system. This type of controller will add additional error states to the system, where the aim is to improve the reference tracking. A more detailed description of the integral control is given in Sec. 3.4.7. Similar to the pre-filter, only the states of the tool-point position should be designed to have a reference tracking ability. Meaning, that only the first rows of the output vector C, which corresponds to the output of P_t, should be used in Eq. 3.125. The augmented state equations describing the system can be written as.

$$\begin{bmatrix} \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & -C_{P_t} \\ 0 & A \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ B \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{4.63}$$

$$y = \begin{bmatrix} 0 & C \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \tag{4.64}$$

Where C_{P_t} corresponds to the first rows of C, which are related to the states of P_t.

As formulated by Eq. 3.129, the feedback law is given by.

$$\begin{aligned} u &= -K_o x + K_e z \\ &= -\begin{bmatrix} -K_e & K_o \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \\ &= -K \begin{bmatrix} z \\ x \end{bmatrix} \end{aligned} \tag{4.65}$$

The feedback gain K can be computed by the method of LQR, due to the added error states $z \in \mathbb{R}^3$, an extension of the weighting matrix for the system state Q_i is needed. The weighting matrix Q_i and the weight matrix for the control effort R_i is given by.

$$Q_i = \begin{bmatrix} Q_{i,z} & 0 & 0 & 0 & 0 \\ 0 & Q_{i,P_t} & 0 & 0 & 0 \\ 0 & 0 & Q_{i,\dot{P}_t} & 0 & 0 \\ 0 & 0 & 0 & Q_{i,\phi} & 0 \\ 0 & 0 & 0 & 0 & Q_{i,\dot{\phi}} \end{bmatrix} \tag{4.66}$$

$$R_i = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{4.67}$$

where the elements of the weight matrix Q_i are given by the following diagonal matrices.

$$\begin{aligned} Q_{i,z} &= diag(\begin{bmatrix} 10^3 & 10^3 & 10^3 \end{bmatrix}) \\ Q_{i,P_t} &= diag(\begin{bmatrix} 10^3 & 10^3 & 10^3 \end{bmatrix}) \\ Q_{i,\dot{P}_t} &= diag(\begin{bmatrix} 10^2 & 10^2 & 10^2 \end{bmatrix}) \\ Q_{i,\phi} &= diag(\begin{bmatrix} 10^4 & 10^4 \end{bmatrix}) \\ Q_{i,\dot{\phi}} &= diag(\begin{bmatrix} 10^3 & 10^3 \end{bmatrix}) \end{aligned} \tag{4.68}$$

The resulting feedback gain K, computed by LQR, yields the following result.

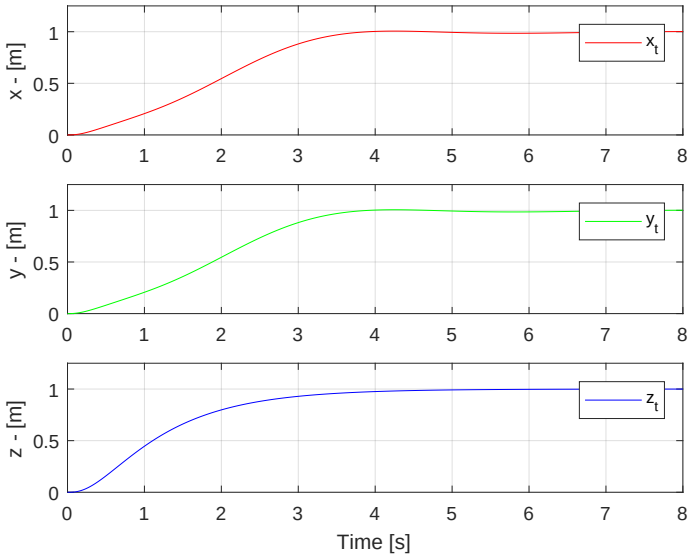
$$K = \begin{bmatrix} -K_e & K_o \end{bmatrix} \tag{4.69}$$

where

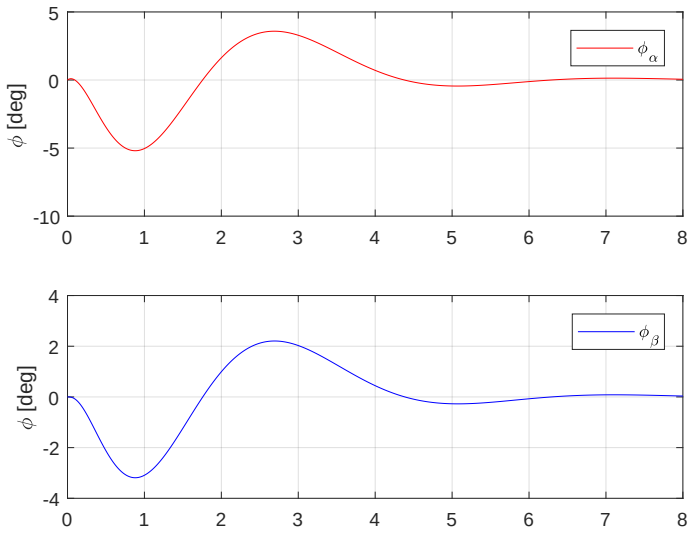
$$-K_e = \begin{bmatrix} 31.62 & 0 & 0 \\ 0 & 31.62 & 0 \\ 0 & 0 & 31.62 \end{bmatrix}$$

$$K_o = \begin{bmatrix} 69.85 & 0 & 0 & 61.32 & 0 & 0 & 241.17 & 0 & -13.45 & 0 \\ 0 & 53.95 & 0 & 0 & 44.43 & 0 & 0 & -71.00 & 0 & 13.27 \\ 0 & 0 & 43.17 & 0 & 0 & 13.65 & 0 & 0 & 0 & 0 \end{bmatrix}$$

A step response of the closed loop system is performed, where a step input is given to each component of the tool-point position vector P_t. The tool-point position response is shown in Fig. 4.33a, and the related response of the pendulum angles are presented in Fig. 4.33b.



(a) Tool-Point Position Response



(b) Pendulum Angle Response

Figure 4.33: Closed Loop Step Response of the Non-Linear Suspended Load and Robot System, with State-Feedback Integral Control

The Simulink model of the non-linear suspended load and robot system, with a state-feedback integral control and extended Kalman filter, is shown in Fig. 4.34.

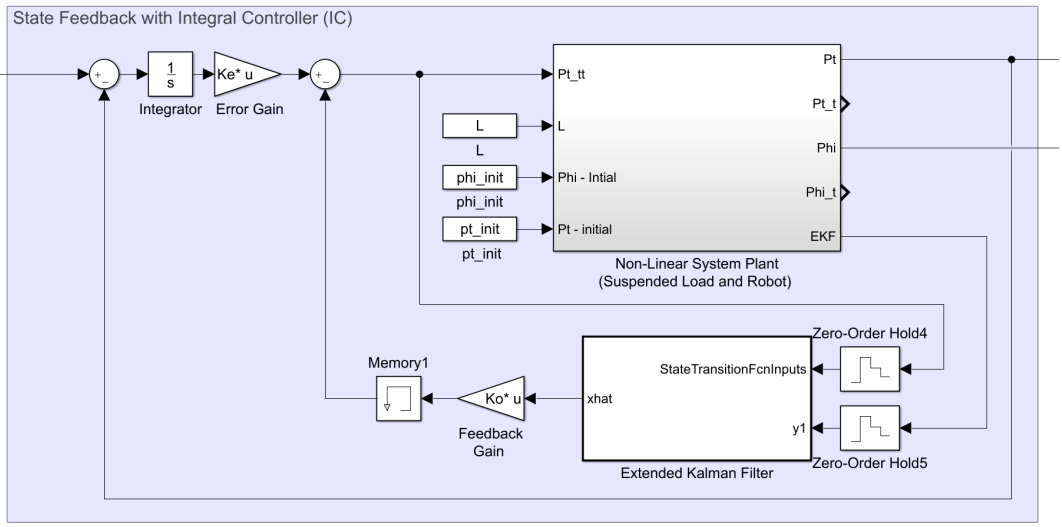


Figure 4.34: Simulink Model of the Non-Linear Suspended Load and Robot System, with State-Feedback Integral Control

The Matlab scripts used to design the functions of the Extended Kalman Filter, and the scripts related to the control system for the state-feedback pre-filter and integral controller are available in App. C.5.

4.5.5 Non-Linear Control

Similar to the control system of the simple pendulum, it is desired to design a non-linear controller for the 3-dimensional suspended load, based on the concept of virtual damping. To compensate for the pendulum angles, a damping effect is added to the undamped system, by manipulation of the input accelerations. A desired dynamics for the pendulum angles are given on the form.

$$\ddot{\phi} + d\dot{\phi} + k\phi = 0 \tag{4.70}$$

Where d is the is the damping coefficient, and $\phi = [\phi_\alpha\,,\,\phi_\beta]$ corresponds to the pendulum angles.

The differential equations describing the dynamics of the 3D pendulum are given by Eq. 3.56 and Eq. 3.57, and by using the earlier defined assumption of a constant wire length $\dot{L}_w = 0$, these equations of motion can be rewritten as.

$$\begin{aligned}\ddot{\phi}_\alpha - \big(&\ddot{x}_t \cdot \cos\phi_\alpha + \ddot{y}_t \cdot sin\phi_\alpha \cdot \sin\phi_\beta \\ &- \ddot{z}_t \cdot \sin\phi_\alpha \cdot \cos\phi_\beta - g \cdot \sin\phi_\alpha \cdot \cos\phi_\beta \\ &- L_w \cdot \dot{\phi}_\beta^2 \cdot \sin\phi_\alpha \cdot \cos\phi_\alpha\big)/L_w = 0 \end{aligned} \tag{4.71}$$

$$\begin{aligned}\ddot{\phi}_\beta - \big(&- \ddot{y}_t \cdot \cos\phi_\beta - \ddot{z}_t \cdot \sin\phi_\beta \\ &- g \cdot \sin\phi_\beta + 2 \cdot L_w \cdot \dot{\phi}_\alpha \cdot \dot{\phi}_\beta \cdot \sin\phi_\alpha\big)/\big(L_w \cdot \cos\phi_\alpha\big) = 0 \end{aligned} \tag{4.72}$$

It can be seen, that the describing equations of motion are a set of coupled differential equations, i.e., the solution of the pendulum angles ϕ_α and ϕ_β are dependent on each other.

Focusing on Eq. 4.71, it is desired to add a part a virtual damper on the form $d_x \cos\dot{\phi}_\alpha$, where d_x is a damping coefficient. Analyzing Eq. 4.71, it can be seen that the only part of the equation not influenced by both angles is the part containing the parameter $\ddot{x}_t$. Hence, a virtual damping effect will be constructed by manipulating this acceleration component. The value of $\ddot{x}_t$ is computed by the following defined relation.

$$\begin{aligned} d_x\dot{\phi}_\alpha &= -\frac{\ddot{x}_t cos\phi_\alpha}{L_w} \\ \Rightarrow \ddot{x}_t &= -\frac{d_x L_w \dot{\phi}_\alpha}{\cos\phi_\alpha} \end{aligned} \tag{4.73}$$

A similar approach is used for the virtual damping of the angle ϕ_β, from Eq. 4.72, where the part containing the acceleration $\ddot{y}_t$ is used to compute the virtual damping $d_y\dot{\phi}_\beta$. The value of $\ddot{y}_t$ is computed as follows.

$$\begin{aligned} d_y\dot{\phi}_\beta &= \frac{\ddot{y}_t t \cos\phi_\beta}{L_w \cos\phi_\alpha} \\ \Rightarrow \ddot{y}_t &= \frac{d_y L_w \dot{\phi}_\beta \cos\phi_\alpha}{\cos\phi_\beta} \end{aligned} \tag{4.74}$$

Implementation of these anti-swing compensators requires inputs for the angular position and velocity of the pendulum angles, where the latter is not considered to be measurable. Hence, the extended Kalman Filter will be used to feed the estimate of these states to the virtual dampers. An updated version of the virtual dampers is given as.

$$\ddot{x}_t = -\frac{d_x L_w \hat{\dot{\phi}}_\alpha}{\cos\hat{\phi}_\alpha} \tag{4.75}$$

$$\ddot{y}_t = \frac{d_y L_w \hat{\dot{\phi}}_\beta \cos\hat{\phi}_\alpha}{\cos\hat{\phi}_\beta} \tag{4.76}$$

Eq. 4.75 and Eq. 4.76 can now be added to the original signal of the acceleration input of $\ddot{x}_t$ and $\ddot{y}_t$, respectively. Which will result in a damping effect of the 3D pendulum angles ϕ_α and ϕ_β.

A figure of the anti-swing system for the non-linear system model can be seen in Fig. 4.35. The non-linear virtual damping controllers are added as Matlab functions based on Eq. 4.75 and Eq. 4.76, where the inputs are obtained from the estimates of the Extended Kalman Filter A cascade controller is added to enable reference tracking of the tool-point x_t and y_t position. The inner velocity loop are controlled by a proportional derivative (PD) controller, and the outer position loops are governed by a proportional integral (PI).

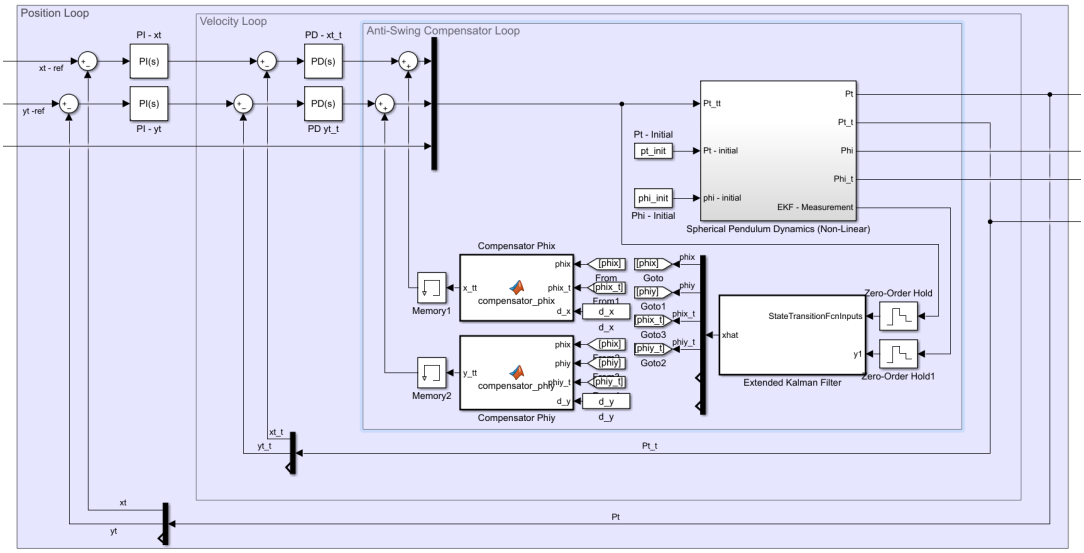


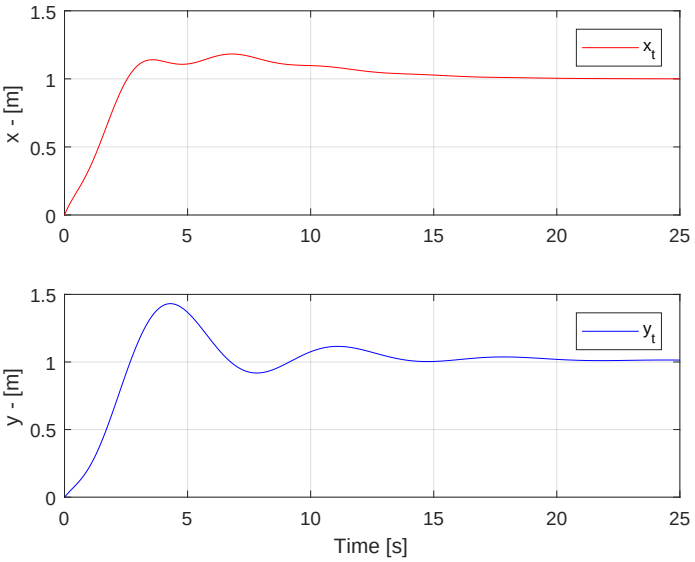
Figure 4.35: Simulink Model of the Non-Linear Suspended Load and Robot System, with Virtual Damper and Cascade Control

No control of the tool-point z_t position is considered for this scenario. The values of the damping coefficients are found by empirical tests, where the final values $d_x = 7$ and $d_y = 5$ are selected. Tuning of the cascade controllers is conducted by an experimental approach, using Matlab's System Identification and PID tuner toolbox. The corresponding values are listed in Tab. 4.5. A closed loop step response of the system is shown in Fig. 4.36, where the response of z_t is omitted due to no control effort is acting on this parameter.

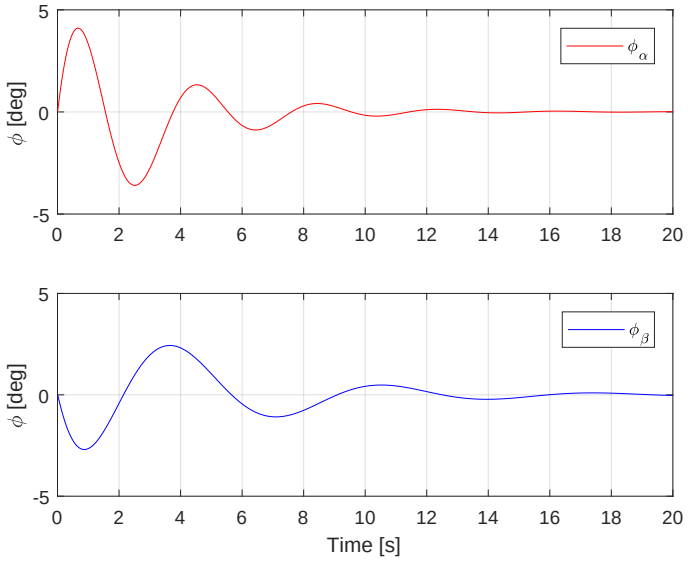
Table 4.5: Cascade Controller Parameters

Velocity Loop		
Parameter	K_p	K_d
$\dot{x}_t$	4.786	3.324
$\dot{y}_t$	1.002	0.408

Position Loop		
Parameter	K_p	K_i
x_t	0.577	0.103
y_t	0.788	0.072



(a) Tool-Point Position Response



(b) Pendulum Angle Response

Figure 4.36: Closed Loop Step Response of the Non-Linear Suspended Load and Robot System, with Virtual Damper and Cascade Control

4.6 Control Design Motion System

This section will present the control design for the anti-swing system of the full motion model described by Sec. 4.3. A control system for the 3-dimensional suspended load system can be established by the methods described by Sec. 4.5, the next step involves extending the system to include the motion induced by the Stewart platform while compensating for the swing-angles. Related to the design of such a controller a few assumptions is made for the full motion system.

- The Comau robot is considered as a rigid system, where no dynamic is included for the joint actuation.
- Feedback of the tool-point position and velocity, together with measurements of the wire length and the suspended load's Euler-angles and velocity are considered to be available.
- Dynamics related to the wire length is not considered in this section, where this problem is assumed to be solvable by an independent controller.
- Measurements of the Stewart platform's relative position and velocity are assumed to be available.

Combining the motion of the Stewart platform with the Comau robot will give a system where the tool-point motion is governed by both the robot joints and the relative motion of the platform. Which leads to an expression of the tool-point motion given relative to the neutral frame of the platform, see Sec. 3.3 for a detailed formulation.

Related to the control design, the tool-point motion relative to the robot base is desired to act as the input to the system. Hence the combined model of the Stewart platform and Comau robot (Fig. 4.14), is combined with the robot's inverse kinematics (Fig. 4.4a). The resulting system is shown in Fig. 4.37, where the input is the tool-point motion given relative to the robot base, and the system output is the wave induced tool-point motion, given relative to the neutral frame of the platform.

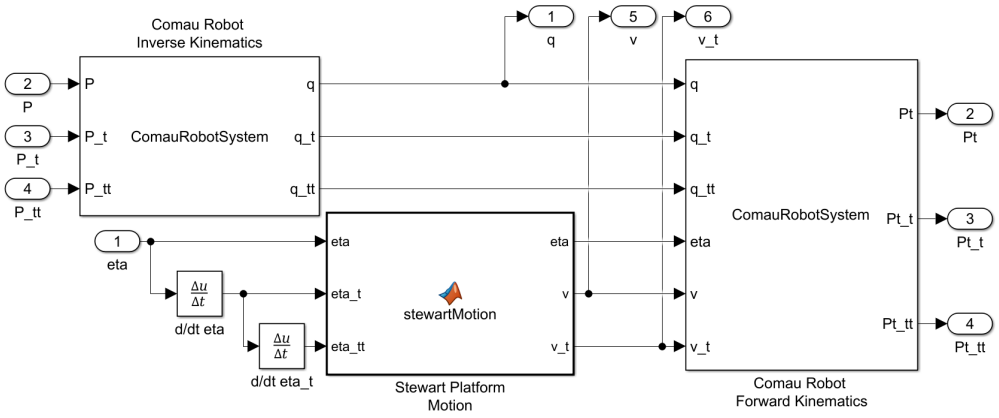


Figure 4.37: System Model of the Comau Robot Kinematics with Stewart Platform Motion

The system used for the 3-dimensional suspended load/pendulum is equivalent to the previously presented system given by Fig. 4.9, where the motion of the 3-dimensional suspended load is directly influenced by the motion of the tool-point and the length of the wire.

4.6.1 Non-Linear System Plant

The non-linear plant for the full motion system, involves the relative motion of the Stewart platform (Sec. 3.3.2), motion kinematics of the Comau robot (Sec. 3.3.3), information of the wire length and related wire rate, as well as the dynamics of the suspended load (Sec. 3.2.2). The aim for the control design, is to develop a system capable of compensating for the Euler-angles of the pendulum, by actuation the robot's tool-point, relative to its respective base frame.

The non-linear system describing the full motion system can be identified as a MIMO system, where the state-vector is defined as.

$$x = \begin{bmatrix} \eta \\ v \\ \dot{v} \\ P_t^r \\ \dot{P}_t^r \\ \ddot{P}_t^r \\ L_w \\ \dot{L}_w \\ \phi \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} \tag{4.77}$$

and the input vector is given by.

$$u = \begin{bmatrix} \ddot{P}_t^r \\ L_w \end{bmatrix} \tag{4.78}$$

Where:

x	-	State vector	$x(t) \in \mathbb{R}^{35}$
u	-	Input vector	$u(t) \in \mathbb{R}^4$
η	-	Stewart platform orientation	$\eta(t) \in \mathbb{R}^6$
v	-	Stewart platform velocity	$v(t) \in \mathbb{R}^6$
$\dot{v}$	-	Stewart platform acceleration	$\dot{v}(t) \in \mathbb{R}^6$
P_t^r	-	Tool-point position	$P_t(t) \in \mathbb{R}^3$
$\dot{P}_t^r$	-	Tool-point velocity	$\dot{P}_t(t) \in \mathbb{R}^3$
$\ddot{P}_t$	-	Tool-point acceleration	$\ddot{P}_t(t) \in \mathbb{R}^3$
L_w	-	Wire length	$L_w(t) \in \mathbb{R}$
$\dot{L}_w$	-	Wire length rate	$\dot{L}_w(t) \in \mathbb{R}$
ϕ	-	Pendulum Euler-angles	$\phi(t) \in \mathbb{R}^2$
$\dot{\phi}$	-	Pendulum Euler-angular velocity	$\dot{\phi} \in \mathbb{R}^2$

As stated by the assumptions, the orientation and velocity of the Stewart platform are considered to be measurable. In a physical implementation, these are obtained from MRU measurements. The feedback of the tool-point position and velocity are also considered to be available, where the values are given relative to the robot base frame.

Information of the wire length is assumed to be obtainable, typically as a feedback signal from the winch. The Euler-angles and related velocities are considered to be measurable. This data recording is assumed to be achievable by employing a vision system, capable of tracking the suspended load, for example by the motion-capture system installed in the motion-lab (Sec. 2.1.5) or by a camera system as illustrated in Fig. 3.8. The Euler-angles can then be computed by the approach described by Sec. 3.3.5

The describing equations for the non-linear motion system can be derived as follows.

$$\dot{x} = f(x,u) = \begin{bmatrix} J(\eta)v \\ \dot{v} \\ \mathbf{0} \\ \dot{P}_t^r \\ \ddot{P}_t^r \\ \mathbf{0} \\ \dot{L}_w \\ \mathbf{0} \\ \dot{\phi} \\ \ddot{\phi} \\ \mathbf{0} \end{bmatrix} \tag{4.79}$$

$$y = h(x) = \begin{bmatrix} \eta \\ v \\ P_t^r \\ \dot{P}_t^r \\ L_w \\ \phi \\ \dot{\phi} \end{bmatrix} \tag{4.80}$$

Where the ship Jacobian $J(\eta)$ is calculated by Eq. 3.67. $\ddot{\phi} = [\ddot{\phi}_\alpha\,,\, \ddot{\phi}_\beta]$ are derived from the equations of motion (Eq. 3.56 and Eq. 3.57), using the updated tool-point acceleration given relative to the Stewart platform, see Eq. 3.82.

4.6.2 Extended Kalman Filter Estimator

To estimate the states of the full motion system, an Extended Kalman Filter is implemented. The non-linear estimator is designed based on the theory presented by Sec. 3.4.11, where the state-transition function $f(x,u)$ are directly derived from Eq. 4.79, and Eq. 4.80 is used as the measurement function. The related Jacobian matrices $F(x,u)$ and $H(x)$ are computed as described by Eq. 3.166 and Eq. 3.167.

Value of the covariance matrices for initial states P, process noise Q and measurement noise R, are found by an empirical approach to yield a satisfactory estimate of the system states. The numerical values for these matrices are listed in Tab. 4.6, where each element of the same state has been assigned the same value.

Table 4.6: Numerical Values of the Covariance Matrices

State	Covariance P	Covariance Q	Covariance R
η	0.001^2	0.001^2	0.01^2
v	0.001^2	0.05^2	0.01^2
$\dot{v}$	0.001^2	0.05^2	0.001^2
P_t^r	0.001^2	0.05^2	0.001^2
$\dot{P}_t^r$	0.001^2	0.05^2	0.001^2
$\ddot{P}_t^r$	0.001^2	0.05^2	0.001^2
L_w	0.001^2	0.001^2	0.001^2
$\dot{L}_w$	0.001^2	0.05^2	0.001^2
ϕ	0.001^2	0.05^2	0.1^2
$\dot{\phi}$	0.001^2	0.1^2	0.1^2
$\ddot{\phi}$	0.001^2	0.1^2	0.001^2

To mimic a physical implementation, some white Gaussian noise has been added to measurements of the platform motion and the suspended load's Euler-angles. Fig. 4.38 and Fig. 4.39 visualizes the performance of the Extended Kalman Filter. For these plots, a sinusoidal wave signal has been assigned as input to each of the elements of the platform orientation vector, η.

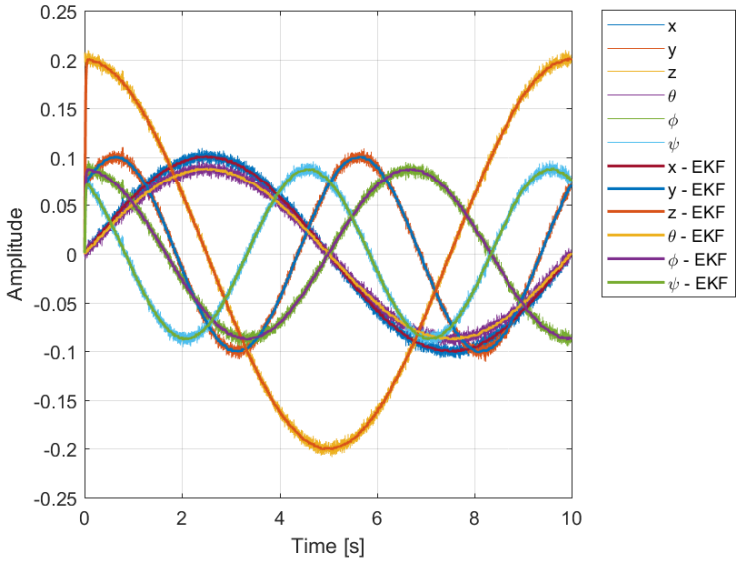


Figure 4.38: Noisy Measurements and EKF Estimation of Stewart Platform Orientation (Relative to Neutral Frame)

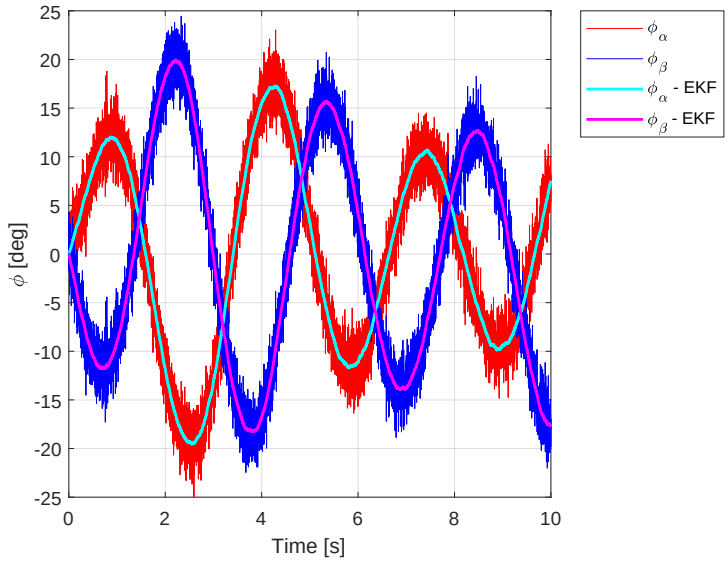


Figure 4.39: Noisy Measurements and EKF Estimation of the Suspended Load's Euler-Angles

The Matlab script used to derive the state-transition function, measurement function, and the related Jacobian matrices for the Extended Kalman Filter can be found in App. C.6.

4.6.3 Linear Control

Compensation of the full motion system, involves the design of a controller capable of actuation of the tool-point while minimizing the swing-angles related to the suspended load. Compared with the system of the suspended load and robot system (Sec. 4.5), the full non-linear motion system has an increased number of states (Eq. 4.77) due to the relative motion induced by the Stewart platform. Also, states of the wire length and rate have been included. However, none of these additional states will be controllable. The design of the linear control of the full motion system will be made similar to the earlier developed controller for the 3-dimensional suspended load system, with only a few modifications. Where the effect of the Stewart platform and winch system, will be treated as plant disturbance.

Hence, the state-vector is given by.

$$x = \begin{bmatrix} P_t^r \\ \dot{P}_t^r \\ \phi \\ \dot{\phi} \end{bmatrix} \tag{4.81}$$

and the input vector is defined as.

$$u = \ddot{P}_t^r \tag{4.82}$$

The describing system equations used to design the linear control, equals to Eq. 4.43 and Eq. 4.44.

$$\dot{x} = f(x,u) = \begin{bmatrix} \dot{P}_t^r \\ u \\ \dot{\phi} \\ \ddot{\phi} \end{bmatrix} \tag{4.83}$$

$$y = h(x) = \begin{bmatrix} P_t^r \\ \phi \end{bmatrix} \tag{4.84}$$

Where $\ddot{\phi} = [\ddot{\phi}_\alpha\,, \ddot{\phi}_\beta]$ are computed using the equations of motion (Eq. 3.56 and Eq. 3.57) with the updated tool-point acceleration, given relative to the Stewart platform (Eq. 3.82). The updated tool-point acceleration requires information of the wire and the Stewart platform orientation, as well as the related velocity and acceleration. Since there are no available states for these parameters in the linear control, a simplification is made, where the wire length is considered to be a constant length of $L_w = 2$, resulting in $\dot{L}_w = 0$, and all the values related to the motion of the Stewart platform are defined to equal zero ($\eta = 0$, $v = 0\,, \dot{v} = 0$).

In Sec. 4.5.3 a linearized version of the non-linear suspended load and robot system was derived, for an equilibrium point defined with the pendulum in a downwards position. For the full system motion the same linearization applies, where the input is set to $u_0 = 0$, and the equilibrium point equals.

$$x_0 = \begin{bmatrix} P_t^r \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{4.85}$$

Calculation of the state-space system matrices, A, B, C, and D evaluated at x_0 and u_0 can be conducted by using Eq. 4.14 - 4.17.

Integral Control

An integral control, described in Sec. 3.4.7, is chosen as the controller for the full motion system. The state-feedback will try to keep the suspended load's Euler-angles as small as possible, and introducing an error state vector, will enable reference tracking for the robot's tool-point. Similar as for the 3-dimensional suspended load control, only the tool-point should be designed to have a reference tracking capability. Hence only the first rows of the output vector C, which are related to the tool-point position, should be used in creating the error states, described by Eq. 3.125. The augmented state equations can be written as.

$$\begin{bmatrix} \dot{z} \\ \dot{x} \end{bmatrix} = \begin{bmatrix} 0 & -C_{P_t^r} \\ 0 & A \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} + \begin{bmatrix} 0 \\ B \end{bmatrix} u + \begin{bmatrix} 0 \\ 1 \end{bmatrix} r \tag{4.86}$$

$$y = \begin{bmatrix} 0 & C \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \tag{4.87}$$

Where z is the error state, and $C_{P_t^r}$ corresponds to the first rows of the output vector, which are related to the output of the tool-point position P_t^r. The feedback law is given by.

$$\begin{aligned} u &= -K_o x + K_e z \\ &= -\begin{bmatrix} -K_e & K_o \end{bmatrix} \begin{bmatrix} z \\ x \end{bmatrix} \\ &= -K \begin{bmatrix} z \\ x \end{bmatrix} \end{aligned} \tag{4.88}$$

Where the feedback gain K for the full motion system is computed by using the method of LQR (Sec. 3.4.5). The weighting matrices for the system state Q_i and for the control effort R_i, is given by.

$$Q_i = \begin{bmatrix} Q_{i,z} & 0 & 0 & 0 & 0 \\ 0 & Q_{i,P_t} & 0 & 0 & 0 \\ 0 & 0 & Q_{i,\dot{P}_t} & 0 & 0 \\ 0 & 0 & 0 & Q_{i,\phi} & 0 \\ 0 & 0 & 0 & 0 & Q_{i,\dot{\phi}} \end{bmatrix} \tag{4.89}$$

$$R_i = \begin{bmatrix} 1 & 1 & 1 \end{bmatrix} \tag{4.90}$$

where the elements of the weight matrix Q_i are given by the following diagonal matrices.

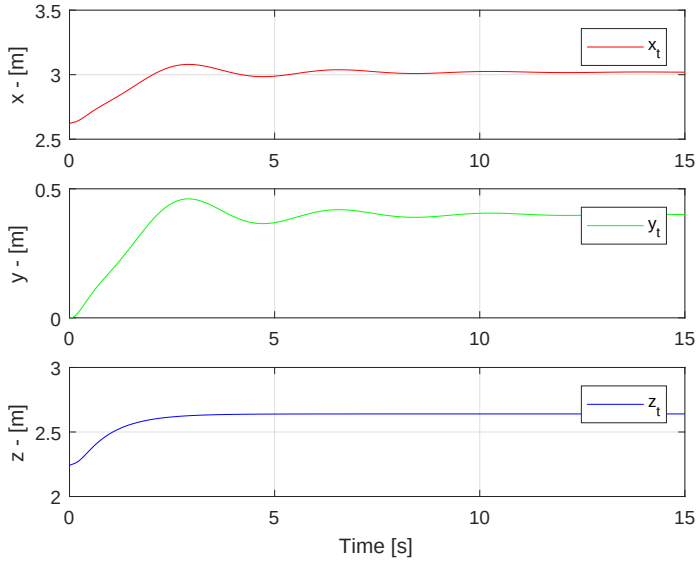
$$\begin{aligned} Q_{i,z} &= diag(\begin{bmatrix} 10^3 & 10^3 & 10^3 \end{bmatrix}) \\ Q_{i,P_t} &= diag(\begin{bmatrix} 10^3 & 10^3 & 10^3 \end{bmatrix}) \\ Q_{i,\dot{P}_t} &= diag(\begin{bmatrix} 10^2 & 10^2 & 10^2 \end{bmatrix}) \\ Q_{i,\phi} &= diag(\begin{bmatrix} 10^4 & 10^4 \end{bmatrix}) \\ Q_{i,\dot{\phi}} &= diag(\begin{bmatrix} 10^3 & 10^3 \end{bmatrix}) \end{aligned} \tag{4.91}$$

The elements of the feedback gain K, defined as $K = [-K_e\, K_o]$, are calculated and contains the following numerical values.

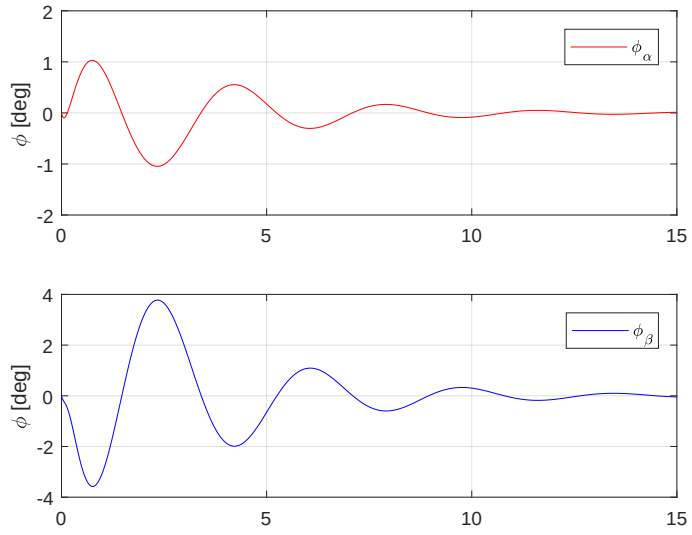
$$-K_e = \begin{bmatrix} 100.00 & 0 & 0 \\ 0 & 100.00 & 0 \\ 0 & 0 & 100.00 \end{bmatrix}$$

$$K_o = \begin{bmatrix} 104.4420 & 0.0003 & 0.1450 & 49.5407 & 0.0002 & 0.1216 & -26.4862 & -46.2177 & 24.6617 & 43.0347 \\ 0.0003 & 104.4420 & -0.1538 & 0.0002 & 49.5407 & -0.1290 & 46.2177 & -26.4863 & -43.0343 & 25.6623 \\ 0.1450 & -0.1538 & 63.3342 & 0.1216 & -0.1290 & 15.0564 & -0.2664 & -0.0639 & 0.2480 & 0.0595 \end{bmatrix}$$

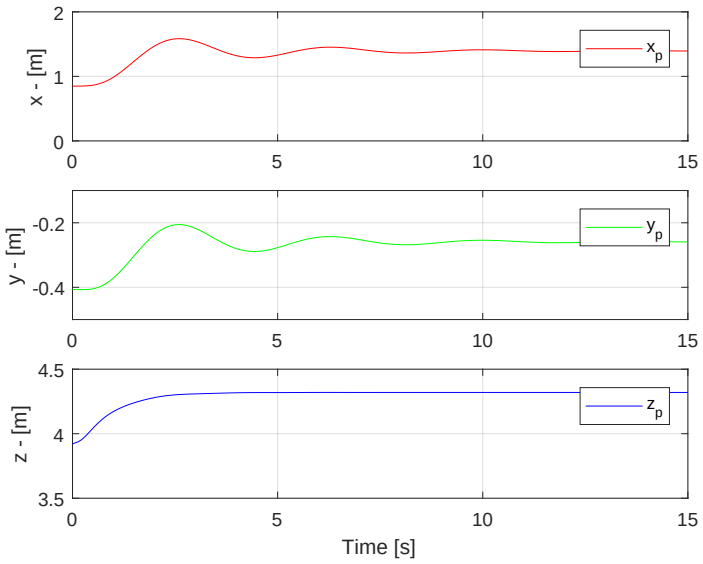
A closed-loop step response is simulated on the system with the state-feedback integral control. The motion of the Stewart platform is set to $\eta = 0$ for this test, and the wire length is $L_w = 2.0$. Since the tool-point is now directly related to the robot kinematics, initial values of the position are equal to the robot's home position, found in Tab. 4.12. A step of 0.4 $[m]$ is used as input to each axis. Fig. 4.40a shows the response of the tool-point position, Fig. 4.40b gives the response of the pendulum Euler-angles, and Fig. 4.40c shows the pendulum position relative to the world coordinate frame.



(a) Tool-Point Position Response (P_t^r)



(b) Suspended Load Euler-Angle Response



(c) Suspended Load Position Response (P_p^g)

Figure 4.40: Closed Loop Step Response of the Non-Linear Motion System, with State-Feedback Integral Control

The Simulink model of the full non-linear motion system with a state-feedback integral control and Extended Kalman Filter can be seen in Fig. 4.41.

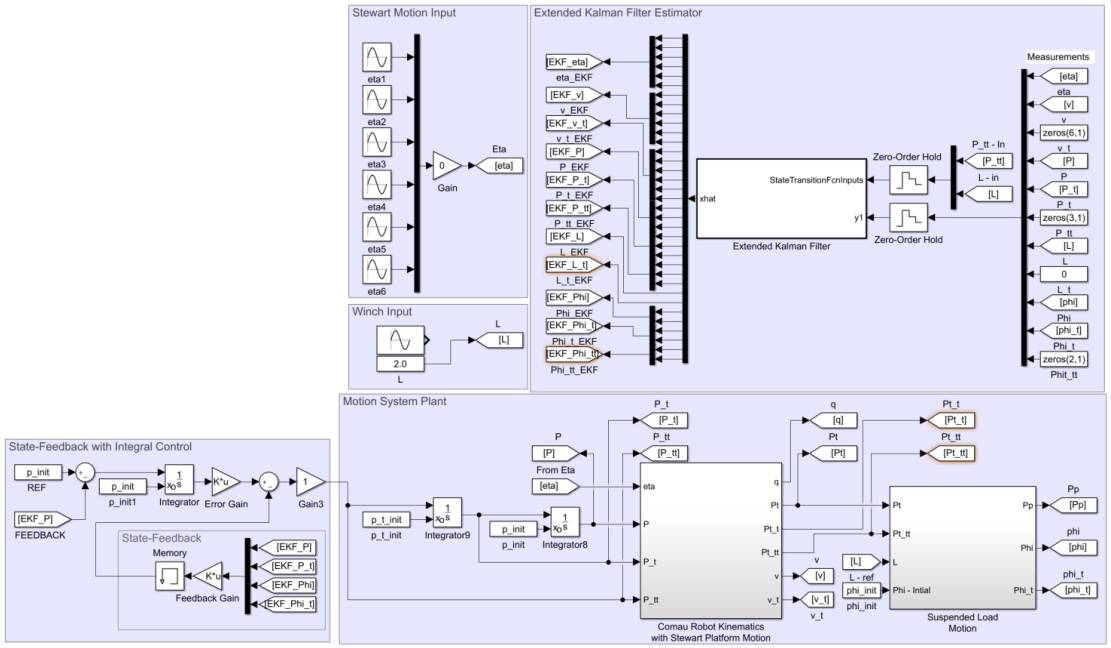


Figure 4.41: Simulink Model of the Non-Linear Motion System, with State-Feedback Integral Control

The Matlab script used to derive the state-feedback integral controller for the motion system is available in App. C.6.

5 Results

In this chapter, the simulation results for developed system models and controller designs will be presented. Here, analyses of different scenarios with different testing parameters are conducted, and a comparison of the systems' performances are made.

5.1 Simulation Inputs

This section will briefly present the different input and reference signals used to test the performance of the developed simulation models and anti-swing controllers. A set of different scenarios are conducted for the tool-point reference and initial position of the suspended load. The generated wave trajectory used to simulated the motion of the Stewart platform will also be presented.

5.1.1 Suspended Load Initial Euler-Angles

The developed simulation models are designed such that the initial values of the suspended load's Euler-angles are required. The simulation results presented in this chapter will consider two different scenarios. One, where the suspended load/pendulum is considered to start in a downwards position, i.e., the values of the initial Euler-angles equals zero. The second scenario will initiate the system with a set of offset angles. The two scenarios are presented by Tab. 5.1 and Tab 5.2.

Table 5.1: Suspended Load Initial Euler-Angles, Downwards Position

Coordinate	**Value**	**Unit**
ϕ_α	0.0	$[deg]$
ϕ_β	0.0	$[deg]$

Table 5.2: Suspended Load Initial Euler-Angles, Offset Position

Coordinate	**Value**	**Unit**
ϕ_α	15.0	$[deg]$
ϕ_β	20.0	$[deg]$

5.1.2 Robot Tool-Point Motion

Three cases are designed as input signals for the position of the reference tracking of the tool-point position. The developed control systems are designed such that the reference signal, should be given relative to the base frame of the robot, P_t^r.

Home-Position

The first scenario considers the robot in a home-position configuration, i.e., no additional position motion is used as input to the robot's tool-point. The home position of the robot, relative to the robot base frame (P_t^r) is obtained from Tab. 4.3, and can be presented as.

Table 5.3: Robot Tool-Point in Home Position Configuration

Coordinate	Value	Unit
x_t^r	2.619	$[m]$
y_t^r	0.0	$[m]$
z_t^r	2.240	$[m]$

Sequence of Step-Input

The second scenario considers a sequence of step signals as a reference input to the tool-point motion. This scenario makes use of the home-position configuration as initial values, and a set of different step inputs are sent to each coordinate of the tool-point position. The plot of this reference signal is shown in Fig. 5.1.

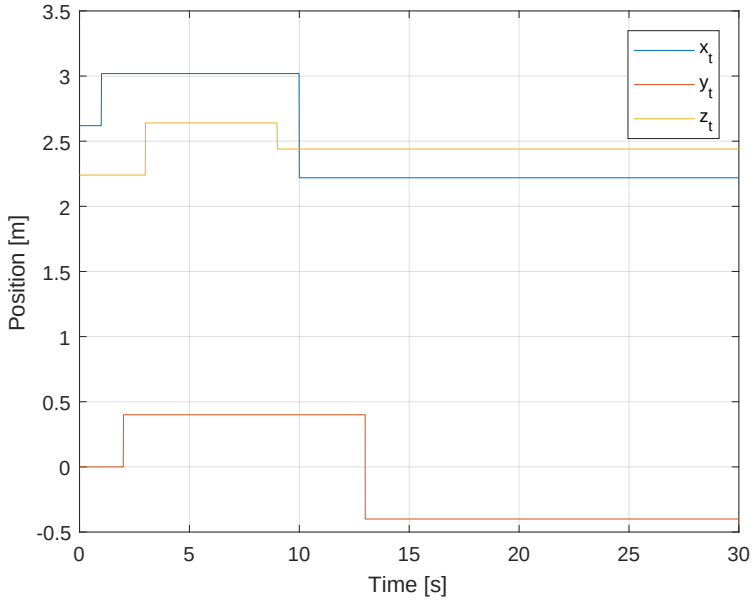


Figure 5.1: Tool-Point Reference Signal, Sequence of Step-Input

Sinusoidal Input

The final scenario for the tool-point's reference signal, involves the use of a sinusoidal input signal to each coordinate of P_t^r. These signals are described by Eq. 5.1, where Tab. 5.4 lists the trajectory parameters. The related plot of the sinusoidal reference signal is shown in Fig. 5.2

$$y = A\sin(2\pi ft) + B \tag{5.1}$$

Where:

y	-	Sinusoidal reference signal	$[m]$
A	-	Amplitude	$[m]$
f	-	Frequency	$[Hz]$
t	-	Time	$[s]$
B	-	Home position bias	$[m]$

Table 5.4: Sinusoidal Reference Trajectory for the Tool-Point Position P_t^r

Parameter	**Amplitude** A $[m]$	**Frequency** f $[Hz]$	**Home-Position** B $[m]$
x_t^r	0.5	0.07	2.619
y_t^r	0.3	0.09	0
z_t^r	0.25	0.15	2.240

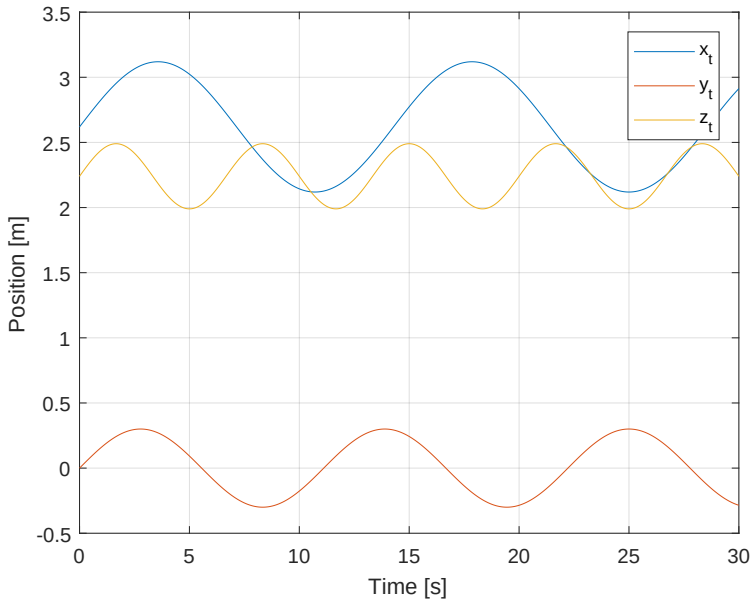


Figure 5.2: Tool-Point Reference Signal, Sinusoidal Input

5.1.3 Stewart Motion

The generated wave motion, simulated by the Stewart platform, is governed by the orientation vector η. For the simulation experiments, the wave trajectory data of Tab. 4.4 is used. A plot of the trajectory can be seen in Fig. 5.3.

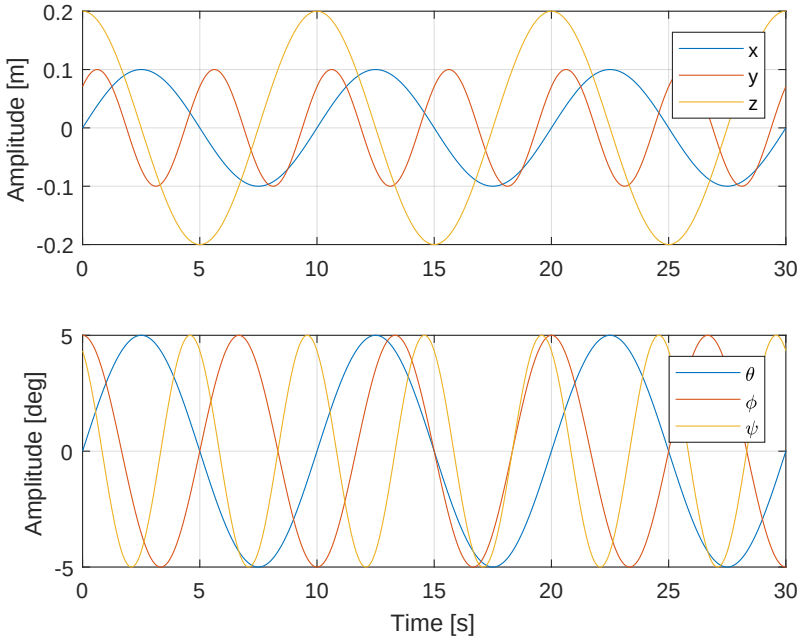


Figure 5.3: Simulated Wave Motion by Stewart Platform

5.1.4 Wire Length

The simulation of the full motion system requires an input value for the wire length. In this thesis, no control system is made for the wire, but the wire length has a direct influence on the suspended load dynamics. For the simulation test, two different scenarios are considered. For most of the simulation experiments, a constant wire length of $L_w = 2.0$ is used. A second scenario uses a sinusoidal signal for the length of the wire, where the signal is given by.

$$L_w = 0.75 \sin(2\pi 0.05) + 2 \tag{5.2}$$

5.2 Simulation Results Suspended Load

In Sec. 4.5 different control system were designed for the 3-dimensional suspended load and robot system. This section will present and compare the simulation results of these control designs. In this model, the suspended load is considered to be attached to the robot's tool-point via a constant wire length of $L_w = 2.0$ $[m]$, and an Extended Kalman Filter is developed to estimate the system states.

A combination of scenarios is designed to test the performance of the derived control systems, where simulation experiments are initiated with different configurations of initial values and signal inputs. Tab. 5.5 describes these scenarios.

Table 5.5: Simulation Scenarios of the 3D Pendulum

Scenario Number	**Tool-Point Reference**	**Pendulum Position**	**Wire Length**
1	Home Position	Offset	$L_w = 2$
2	Step-Input	Downwards	$L_w = 2$
3	Sinusoidal	Downwards	$L_w = 2$
2	Step-Input	Offset	$L_w = 6$

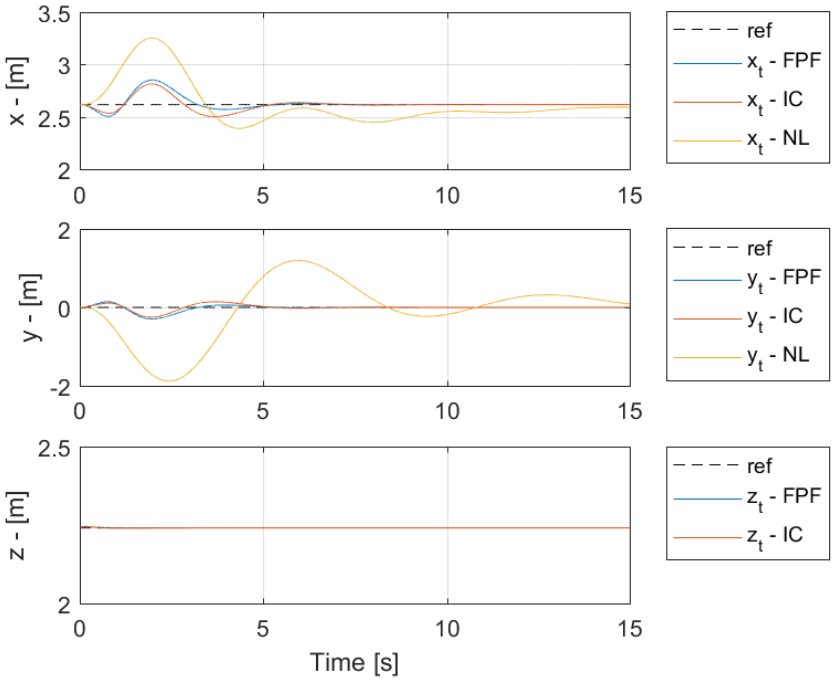
The abbreviations used for the upcoming result plots are listed in Tab. 5.6. It should also be noted that no response of the z_t coordinate is available for the non-linear control system, as described by Sec. 4.5.5.

Table 5.6: Abbreviations used to describe the 3D Pendulum Results

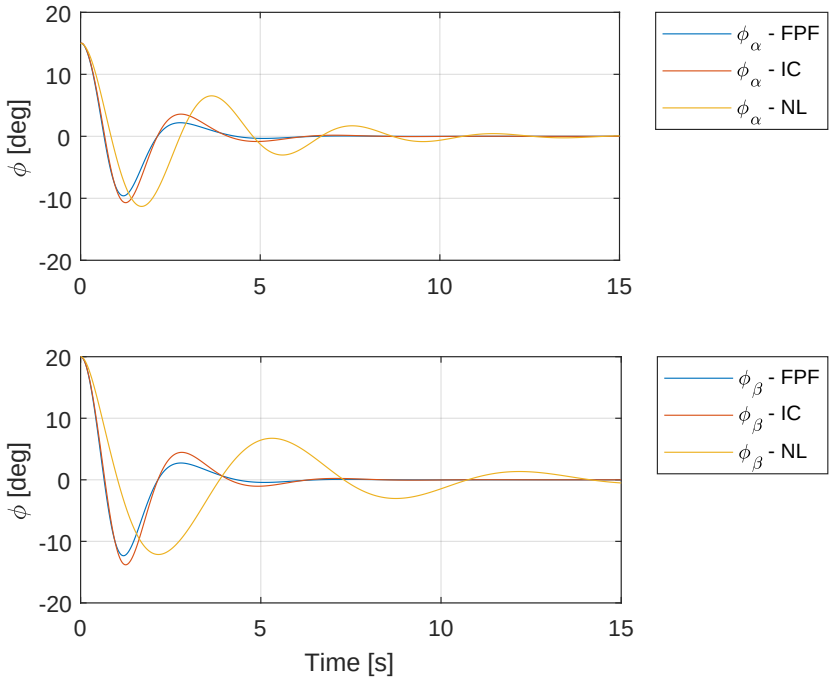
Description	**Abbreviation**
Reference	ref
State-Feedback Pre-Filter Control	FPF
State-Feedback Integral Control	IC
Non-Linear Control	NL

5.2.1 Scenario 1

The first scenario simulates the 3-dimensional suspended payload with initial angle offsets. Here, the tool-point position is given a constant reference input equal to the home position of the robot. Hence, these simulations will give an indication of the controllers ability to compensate for the pendulum offset angles and return to a steady-state. Fig. 5.4a shows the response of the tool-point position for the different control system, and the angular response related to the suspended load is given by Fig. 5.4b.



(a) Tool-Point Position

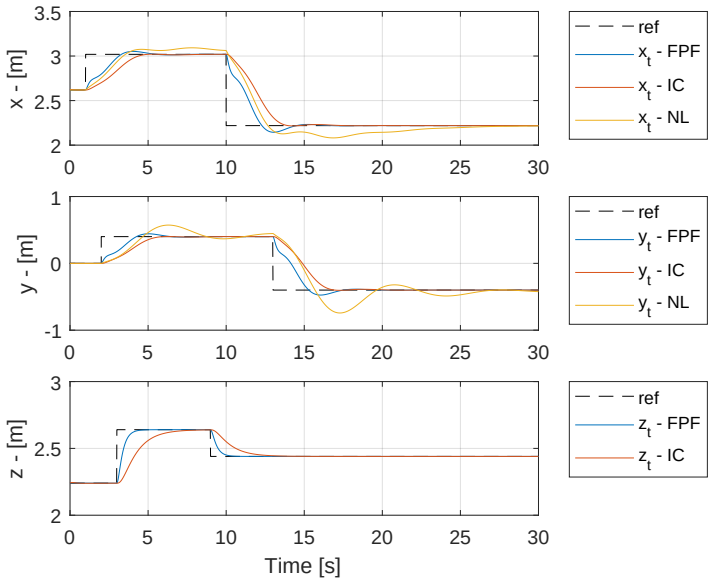


(b) Suspended Load Angles

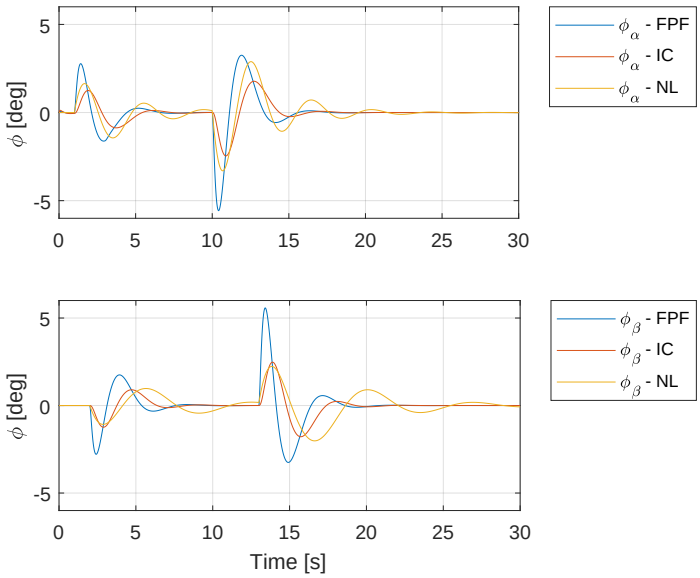
Figure 5.4: Suspended Load Results for Scenario 1

5.2.2 Scenario 2

In scenario 2, the suspended load is initialized in a downwards position, and the tool-point position is given a sequence of step-inputs, as described by Sec. 5.1.2. The simulation results of the different control systems are presented by Fig. 5.5.



(a) Tool-Point Position

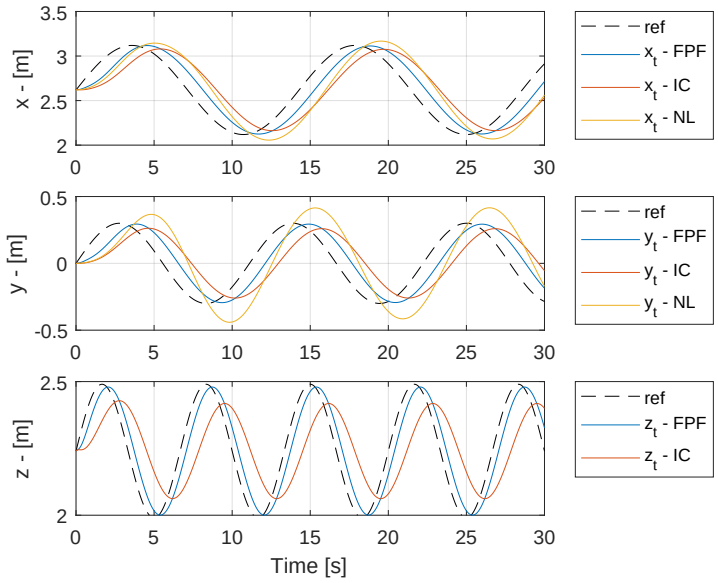


(b) Suspended Load Angles

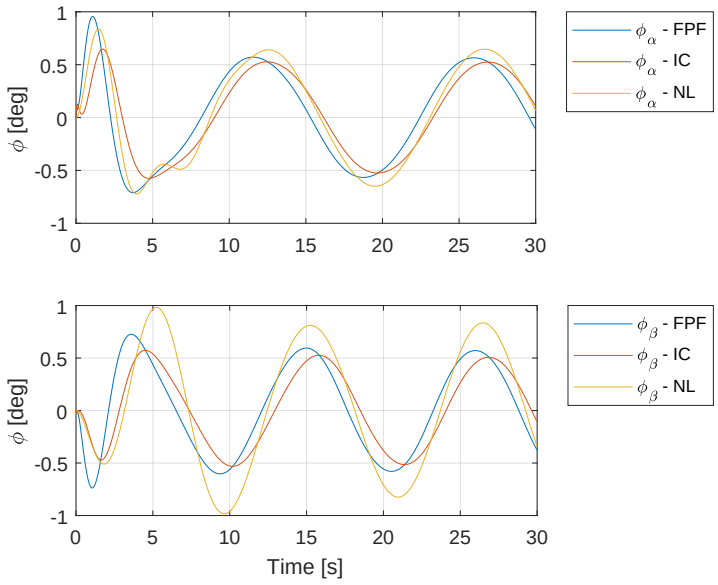
Figure 5.5: Suspended Load Results for Scenario 2

5.2.3 Scenario 3

Scenario 3, maintains the downwards configuration as an initial position for the suspended load. The tool-point position is now governed by sinusoidal reference input. The related simulation response is presented by Fig. 5.6.



(a) Tool-Point Position

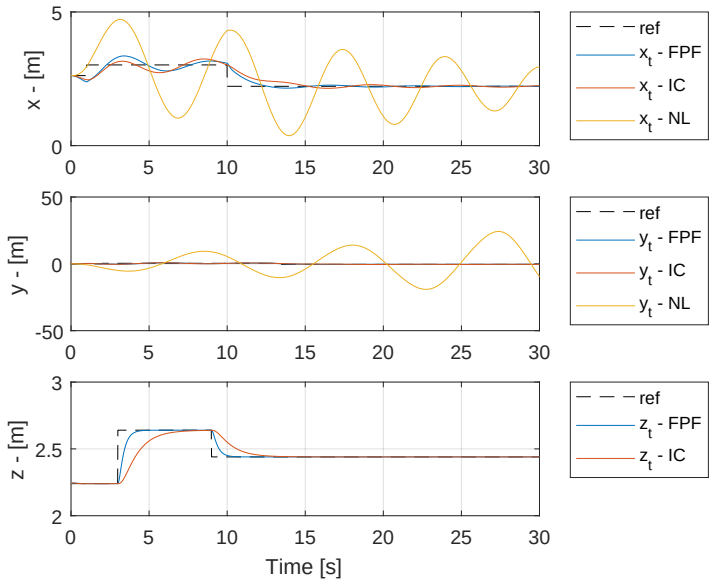


(b) Suspended Load Angles

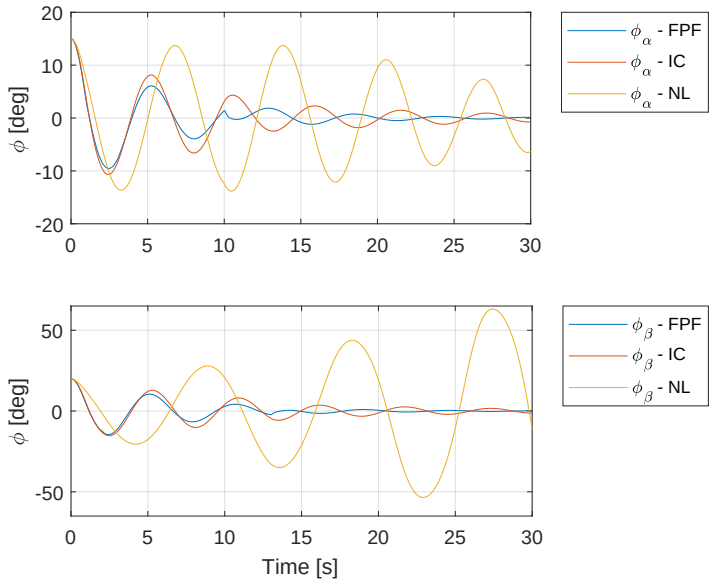
Figure 5.6: Suspended Load Results for Scenario 3

5.2.4 Scenario 4

In scenario 4, the tool-point position is given a sequence of step-inputs as reference signals, and the suspended load has an angular offset. In addition to this, a new wire length of $L_w = 6$ is introduced to the system. The response of this simulation can be seen in Fig. 5.7.



(a) Tool-Point Position



(b) Suspended Load Angles

Figure 5.7: Suspended Load Results for Scenario 4

5.3 Simulation Results Motion System

In Sec. 4.3.1 and Sec. 4.6 a simulation model and a control system for the full motion system was developed. Here, the 3-dimensional suspended load is attached to the tool-point of the Comau robot, where the base of the robot is experiencing relative motion simulated by the Stewart platform. An Extended Kalman Filter is designed to estimate the states of the full system, and a linear state-feedback with integral control is implemented as the control system.

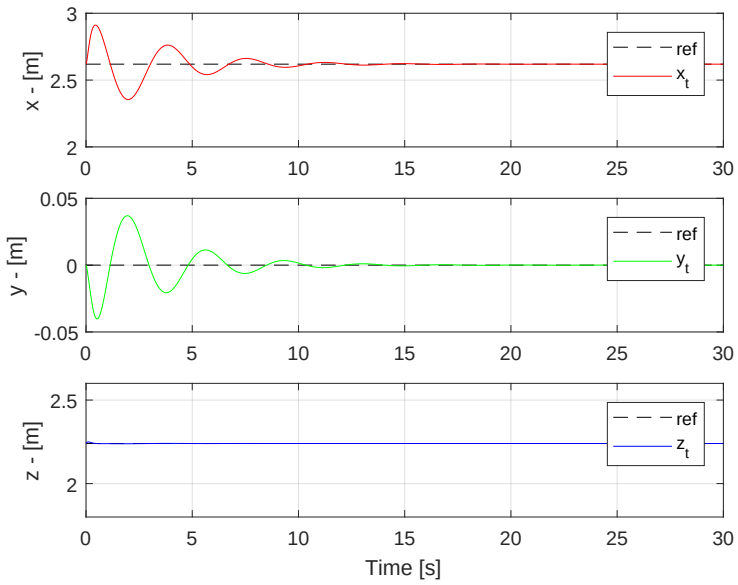
Testing the performance of the control system, different configurations of simulation inputs and initial values (see Sec. 5.1) are introduced to the system. Tab. 5.7 lists the different scenarios used to obtain these simulation results.

Table 5.7: Simulation Scenarios of the Full Motion System

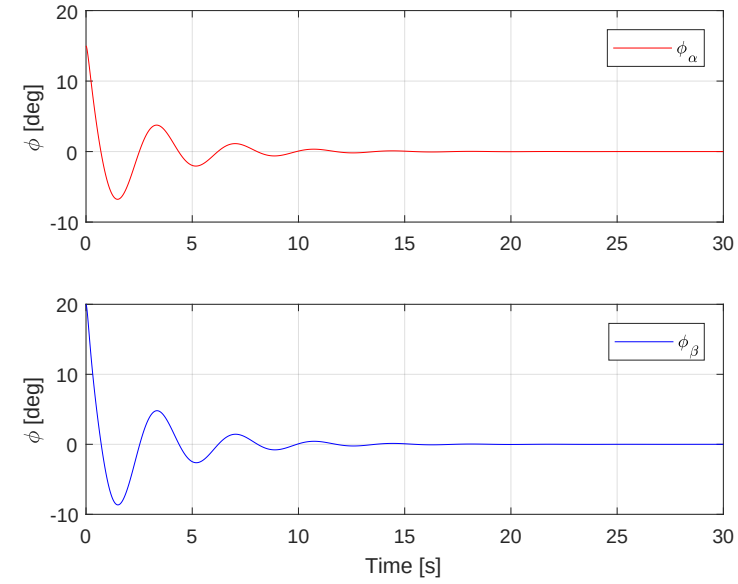
Scenario Number	**Active Controller**	**Tool-Point Reference**	**Platform Motion**	**Pendulum Position**	**Wire Length**
1	Yes	Home Position	No	Offset	$L_w = 2$
2	Yes	Step-Inputs	No	Offset	$L_w = 2$
3	Yes	Sinusoidal	No	Downwards	$L_w = 2$
4	No	Home Position	Yes	Downwards	$L_w = 2$
5	Yes	Home Position	Yes	Downwards	$L_w = 2$
6	Yes	Step-Inputs	Yes	Offset	Sinusoidal

5.3.1 Scenario 1

The first scenario simulates the suspended load with an initial offset configuration, see Tab. 5.2. The reference input of the tool-point position is to settle at the home-position. Fig. 5.8a shows the response of the tool-point position, and Fig. 5.8b plots the angles related to the suspended load. The position of the tool-point is given relative to the base frame of the robot ($\{b_r\}$ in Fig. 3.8), which will also be the case for the upcoming simulation scenarios.



(a) Tool-Point Position (P_t^r)

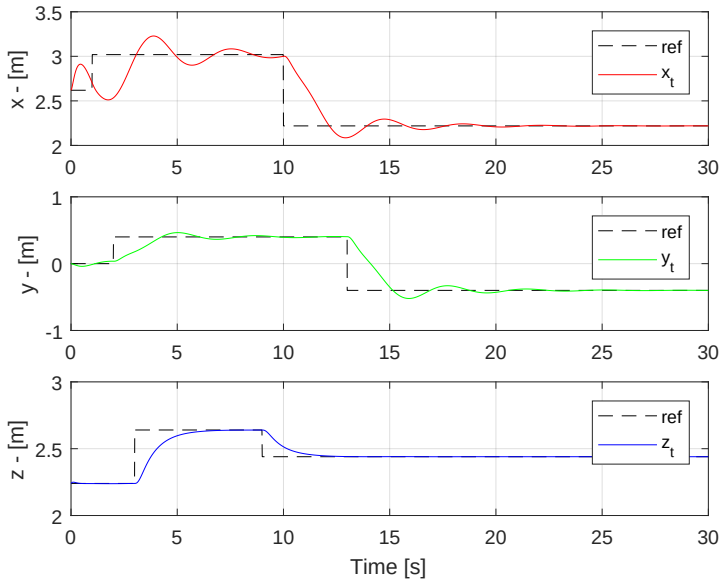


(b) Suspended Load Angles

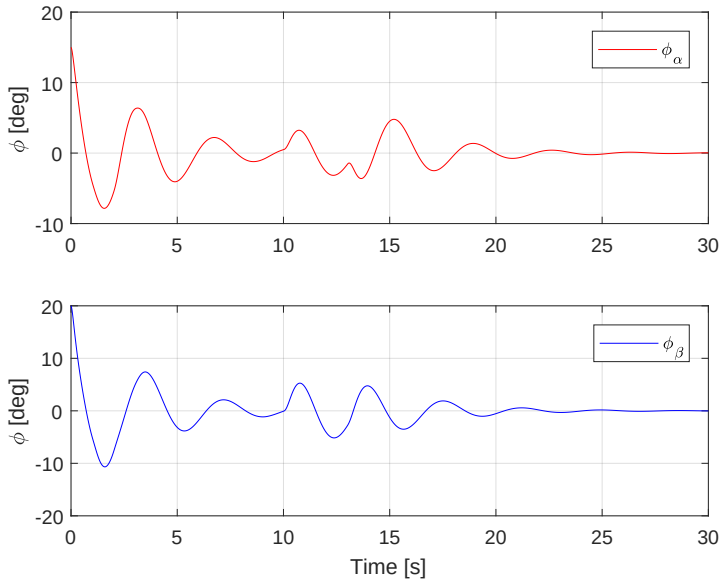
Figure 5.8: Motion System Results for Scenario 1

5.3.2 Scenario 2

In Scenario 2, the suspended load is initiated with an offset position, and the tool-point position is given a reference trajectory equal to the earlier described scenario of a step-input sequence (Sec. 5.1.2). The response of the tool-point position with the reference trajectory can be seen in Fig. 5.9a, and the response of the pendulum angles is given in Fig. 5.9b.



(a) Tool-Point Position (P_t^r)

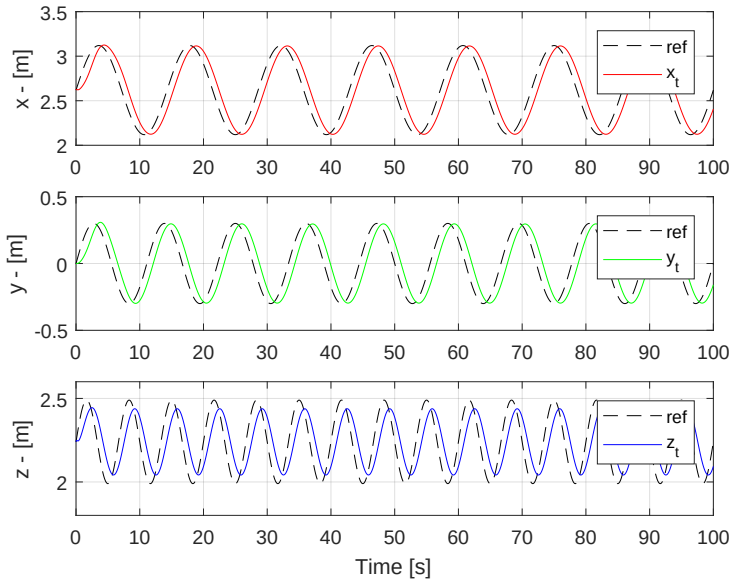


(b) Suspended Load Angles

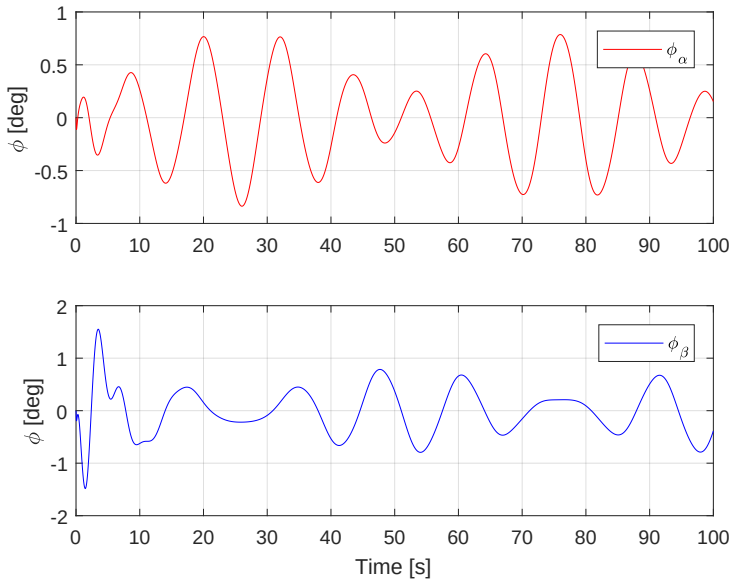
Figure 5.9: Motion System Results for Scenario 2

5.3.3 Scenario 3

Scenario 3, starts with the pendulum at rest, in a downwards position. A sinusoidal reference signal is given to the tool-point position, as described in Sec. 5.1.2. The response of the tool-point position and the suspended load's angles are presented by Fig. 5.10a and Fig. 5.10b, respectively.



(a) Tool-Point Position (P_t^r)

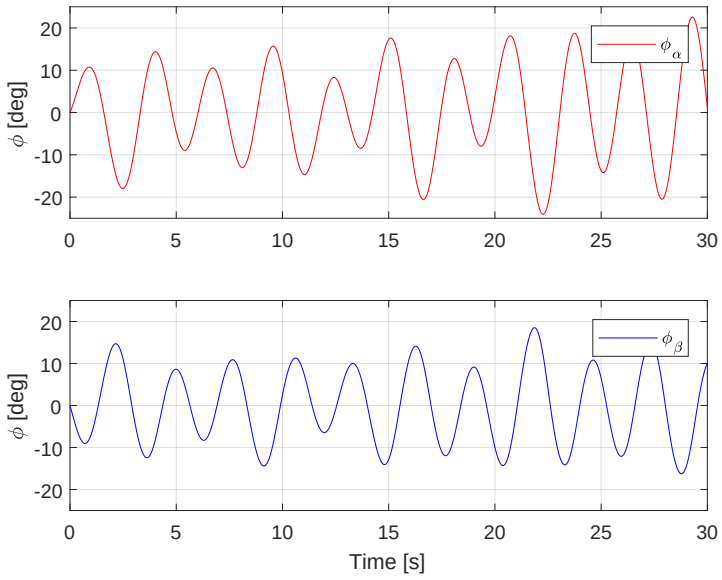


(b) Suspended Load Angles

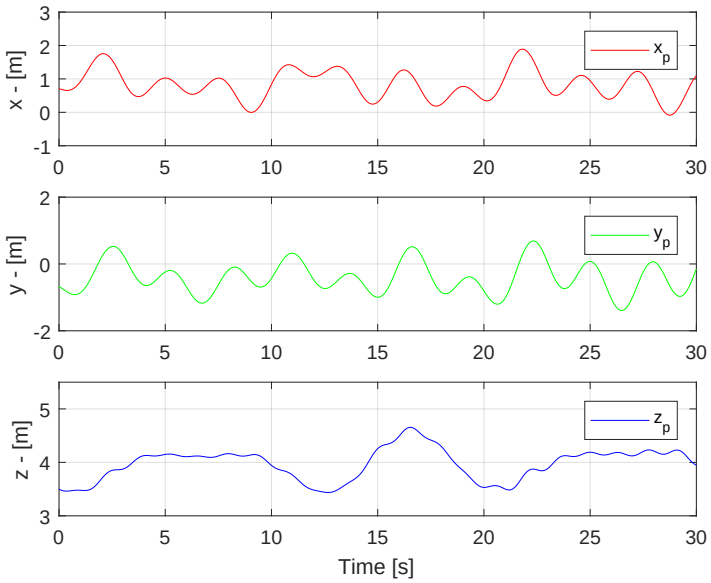
Figure 5.10: Motion System Results for Scenario 3

5.3.4 Scenario 4

In scenario 4, the Stewart platform motion is enabled, and no control system is active. This test will serve as a benchmark, for the future simulation results. Here the suspended load, starting in a downwards position, is disturbed by the generated wave motion. Fig. 5.11a shows the response of the angle associated with the suspended load and Fig. 5.11b shows the position of the suspended load, relative to the world coordinate frame.



(a) Suspended Load Angles

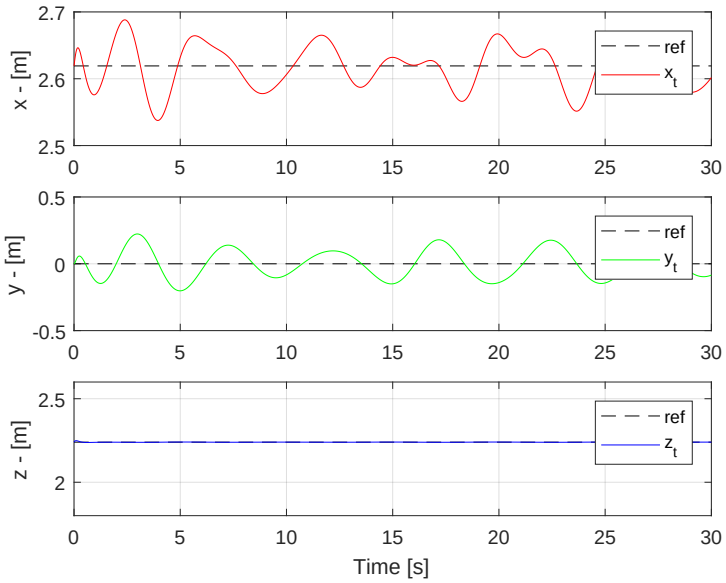


(b) Suspended Load Position (P_p^g)

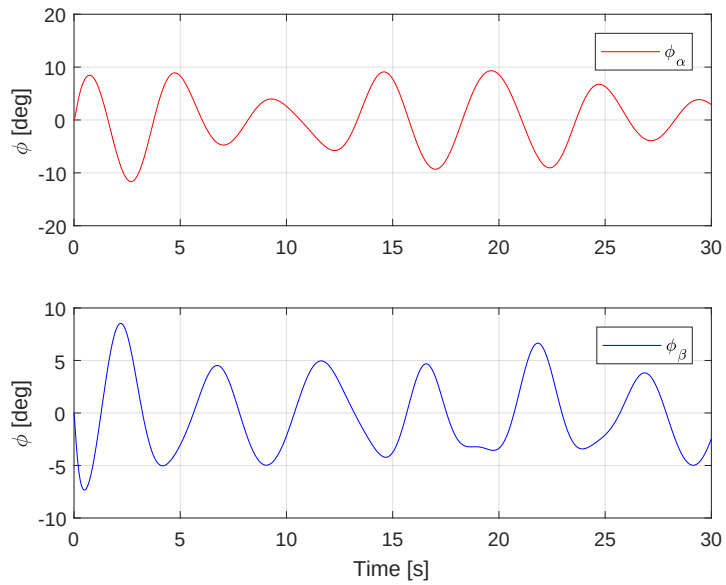
Figure 5.11: Motion System Results for Scenario 4

5.3.5 Scenario 5

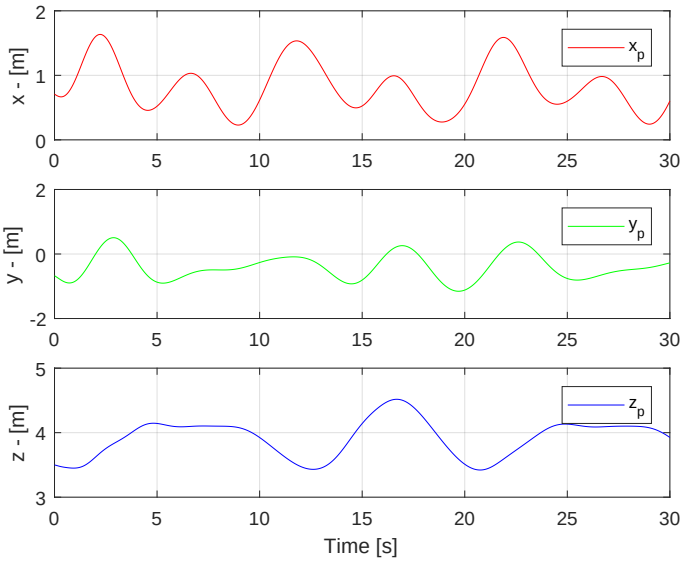
Scenario 5, resembles the previous scenario. The Stewart platform motion is enabled, and the suspended payload starts in a downwards position. Now the control system is activated, where it aims to reduce the swing-angle shown in Fig. 5.11a, and the reference signal for the tool-point is set for the home position. Fig. 5.12a shows the response of the tool-point position, the angular response of the suspended load is given by Fig. 5.12b, and the suspended load position relative to the world coordinate frame is shown in Fig. 5.12c.



(a) Tool-Point Position (P_t^r)



(b) Suspended Load Angles

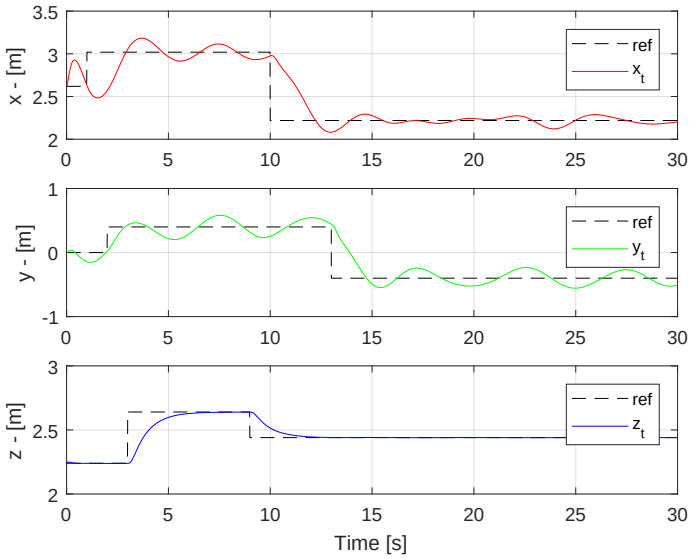


(c) Suspended Load Position (P_p^g)

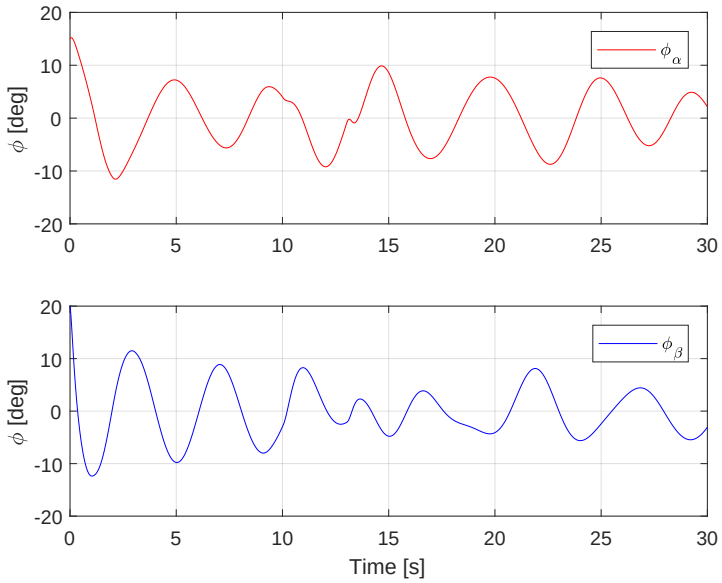
Figure 5.12: Motion System Results for Scenario 5

5.3.6 Scenario 6

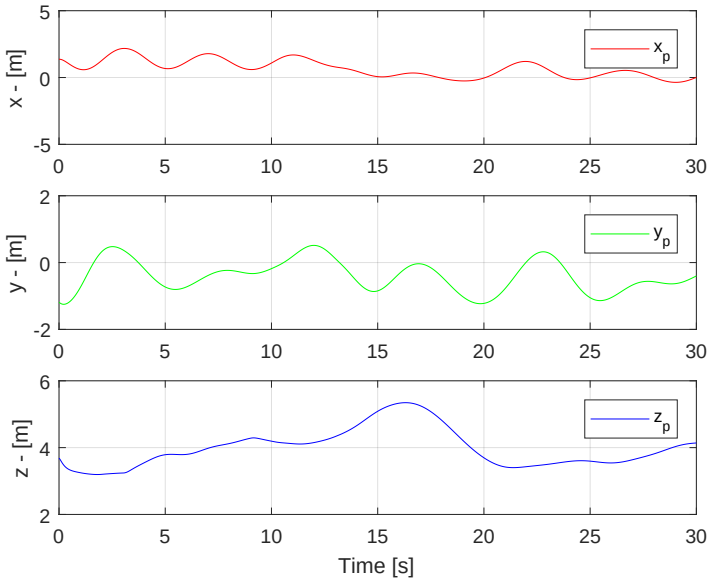
Fig. 5.13, shows the response of the scenario where the tool-point position is given a set of step-inputs, and the load is initiated with an offset position. The motion of the Stewart platform is active, and a sinusoidal input is given to the wire length (Sec. 5.1.4).



(a) Tool-Point Position (P_t^r)



(b) Suspended Load Angles



(c) Suspended Load Position (P_p^g)

Figure 5.13: Motion System Results for Scenario 6

6 Discussion

In this chapter, the most significant methods and results are compared and discussed, together with suggestions for further work and improvement. The primary objective of this thesis is to develop an anti-swing system for a suspended load attached to an industrial robot, which is experiencing a relative motion due to 6-DOF simulated wave motion.

6.1 System Models

Industrial Robot

The modelling of the Comau robot uses the simplification of treating the robot as a rigid system, which leads to an optimal response when assigned with control values. In a real scenario, this is not true, where both joint stiffness and control delay will influence the dynamics of the Comau robot. A suggestion for further work is to conduct an analysis of the physical robot and implement the dynamics into the model. A possible solution is to introduce Eq. 6.1, to add a dynamic effect on the robot's joints.

$$\ddot{q} + 2\zeta_r \omega_r \dot{q} + \omega_r^2 q = K_r \omega_r^2 q_{ref} \tag{6.1}$$

where:

q	-	Joint vector angular position
$\dot{q}$	-	Joint vector angular velocity
$\ddot{q}$	-	Joint vector angular acceleration
q_{ref}	-	Joint vector reference position
ω_r	-	Joint vector's natural frequency
ζ_r	-	Joint vector's damping ratio

Suspended Load

As described in Sec. 3.2.2, the mathematical model of the suspended load assumes the wire to be a massless rigid rod. In a physical implementation, this is not true, where the wire itself will have a mass, which will increase and decrease as the length of the wire varies. The model can be improved by treating the wire as an evenly distributed mass. Which is also the case for the payload, which is considered as a point mass, but in reality, this mass will have a mass moment of inertia. The elongation and deflection of the wire is a relatively complex problem to model, hence the simplification of neglecting these features and treating the wire as a rigid rod. The system of the suspended load is considered as a frictionless system, which will force the system into an everlasting oscillation if exposed to an external force. In a real scenario, the air drag and friction between the wire and tool-point will cause a damping effect on the system. These effects are small, hence the simplification of neglecting them, but adding a small friction coefficient could improve the resemblance between the model and real-life system.

6.2 Control System Design

Simple Pendulum System

Implementation of an anti-swing control system for the 2-dimensional simple pendulum was conducted to compare the estimator performance of a Kalman Filter and an Extended Kalman Filter. The results of Fig. 4.21 showed that by changing parameters of the system plant, the Kalman Filter failed to yield a satisfactory estimation of the system states. Hence the decision was made to use the Extended Kalman Filter for the control design of the 3-dimensional systems.

Another feature tested on the simple pendulum system was the non-linear controller which was based on the concept of virtual damping. Here the acceleration input was manipulated to act as a damping effect on the system plant. An adequate anti-swing compensation was achieved when the system was exposed for a step input. However, a considerably faster and more robust response was given by both the linear state-feedback with pre-filter and by the state-feedback integral controller. An experimental approach was conducted to tune the non-linear control. Further work with a focus on other tuning methods, could indicate if this type of controller has the possibility to behave more responsively and robustly.

Suspended Load System

For the 3-dimensional suspended load system, the actuation of the robot tool-point was included. Here, three different control designs were implemented. Two linear controllers, which were based on the linearization of the non-linear plant, and one non-linear controller, which used the concept of virtual damping. For the latter, a cascade controller was implemented to introduce reference tracking of the tool-point. A system identification of the anti-swing compensation plant were conducted on the system in steady-state, and the values assigned for the inner and outer loop was obtained via empirical methods. Hence the control parameters used for the cascade controller were based on the steady-state of the highly non-linear plant. Further work should be made for the analysis of the virtual damper, whereby using an experimental tuning approach, the system had a tendency to become unstable.

Motion System

Implementing the platform motion to the 3-dimensional suspended load and robot system, introduced several new states to the system, where many of these were uncontrollable. Hence, consideration was made to treat these new states of the platform motion and varying wire length as system disturbance. This consideration enabled the suspended load and robot system's state-feedback integral control to be implemented as a linear controller for the full motion system.

6.3 Simulation Results

Measurements of the Swing Angles

Throughout this thesis, position measurements of the suspended load swing angles are assumed to be available. In a real scenario, these values either has to be measured directly by a sensor system installed at the connection point between the robot's tool-point and wire [21], or by an indirect method of measuring the suspended load's position relative to the Stewart platform. For further work, a measurement system of these angles should be implemented. The Motion-Lab has several available and relevant types of measuring equipment, where a laser tracker and marker, or a vision system can be used to both detect and track the suspended load during operation. Implementation of such a system will introduce noise to the measurements. Fig. 4.39 illustrates an example of such a scenario where an Extended Kalman Filter is used to filter this kind of noise.

Wave Motion

The generated wave motion simulated by the Stewart platform is based upon a set of sinusoidal signals, as shown by Fig. 5.3. A better approximation for the real behavior of a stochastic wave motion could be implemented by using Pierson-Moskowitz Spectrum [27]. Measurements of the Stewart platform motion is estimated based on the generated wave input, which is made as a simplification. Whereas an implementation to the Motion-Lab or a real vessel configuration, this data is only available through the measurements of the MRU, which will contain sensor noise. Fig. 4.38, shows how an Extended Kalman Filter can filter this kind of noise, but for further simulations, this noise is not implemented.

Suspended Load System

Multiple scenarios were simulated for the different control designs of the suspended load and robot system. A recurring result can be seen, where the linear controllers give both a better compensation of the pendulum angles and a preferred reference tracking of the tool-point. In the fourth scenario, when the system plant is changed, the non-linear control is on the verge of instability. A closer look at the overall results of the linear controllers, shows that the pre-filter control yields a faster response for the reference tracking, but integral control yields a superior anti-swing compensation. Since the latter is considered to be most important, this type of controller is chosen for the full motion system.

Motion System

For the multiple scenarios simulated on the full motion system, the anti-swing system proves to be both capable of minimizing the swing angles and to adequately follow a reference signal. For the situation without any control system, presented by the fourth scenario, it can be seen that the swing angles, and load position has a higher oscillation frequency and a larger amplitude than for scenario five, where the control system is active.

6.4 Implementation to Motion-Lab

This thesis has presented different control systems capable of minimizing the swing angles, for scenarios where the industrial robot and suspended load is exposed to wave motion. However, these results are only based on mathematical models and simulation. Further work lies in implementing the developed control system to the physical system available in the Motion-Lab. Instead of using simulated wave trajectories, measurements from MRU should provide the readings of the relative motion, and a system capable of measuring the suspended load angles should be developed and implemented.

The controllers used for the system simulations considers the response of the actuator as infinite fast. However, for a physical implementation the dynamics of the robot will be influential, hence the response time of a physical controller is expected to be slower than the presented simulation results.

7 Conclusion

This thesis addresses the problem of anti-swing compensation for a system consisting of a suspended load attached to an industrial robot, where the base of the robot is experiencing wave-induced motion simulated by a motion platform.

Kinematic analysis is performed for the industrial robot, where the robot is simulated to act as 3-DOF offshore loader crane. A study of the kinematics and dynamics related to the 3-dimensional suspended load is investigated, where the Euler-Lagrange equation derives the equations of motion. Combining the suspended load and the industrial robot, with the 6-DOF wave motion simulated by the motion platform, yields a full motion system. Full system kinematics and relative motion are identified, by use of homogeneous transformations, and derivations of the platform's relative velocities and accelerations.

A set of simulations models, based on the mathematical derivations of the motion systems, are developed in a Matlab Simulink environment. These models allow for a full simulation of the combined motion system, where the suspended load is influenced by the actuation of the robot's tool-point, where the base of the robot will experience motion due to change of platform orientation. The 6-DOF wave motion simulated by the platform is based on a set of sinusoidal signals. A 3D-animation feature is added to visualize the motion of the equipment relative to a world coordinate frame.

Several control systems are developed and implemented to the system model of the suspended load and robot system. For the non-linear system plant, only the position of the robot's tool-point and the swing angles of the 3D pendulum are considered to be measurable. Hence, an Extended Kalman Filter is constructed to estimate the states of the non-linear system. A state-space linearization is performed on the non-linear system plant, where a linear approximation of the system is computed around the equilibrium point of the suspended load in a downwards position. The linearized system is further used to design the two linear controllers. A state-feedback pre-filter control is developed, where the use of the LQR method computes the feedback gain, and a pre-filter gain is added to enable reference tracking of the robot's tool-point. A state-feedback integral control is also developed for the linear plant. Here, additional state-errors are added to the system, where the feedback and integral gain are computed by the method of LQR. A final non-linear controller was also developed for the suspended load and robot system. This controller is based on the concept of virtual damping, where the tool-point acceleration was manipulated to give a damping effect to the original undamped system. A cascade controller was implemented to enable reference tracking of the tool-point, where the tuning parameters were found through an experimental approach. The non-linear control system was merely implemented to investigate the concept of virtual damping, and to compare the response with the linear control designs.

Simulation of the suspended load and robot system with the implemented controllers yielded satisfactory results in anti-swing compensation and tool-point reference tracking. It was shown that the linear controllers outperformed the non-linear control, in both reference tracking and swing compensation. A closer analysis revealed the pre-filter control as preferable in terms of reference tracking, but the integral control gave a more robust compensation of the pendulum angles. It was decided that the latter case was favorable. Hence this type of integral control was used as a basis for the full motion system.

Combining the suspended load and robot system with the platform motion introduces several new states to the non-linear system. An Extended Kalman Filter is implemented as an estimator for the states of the full system with the added platform motion. The introduction of the new states, results in a system where the state-vector is no longer fully controllable. With the only available actuation being the motion of the robot's tool-point, it is not possible to control the states of the relative motion. Compensation of the swing-angles and enabling reference tracking of the robot's tool-point, is performed by the same procedure of linearization as for the suspended load and robot system. A linear system model is constructed of only the robot and suspended load system, where the relative motion and varying wire length is treated as plant disturbance. The dynamics of the 3D pendulum is updated to be influenced by the tool-point motion relative to the platform frame, and control of the tool-point is performed relative to the robot base frame. This design yields a system where it is possible to implement the state-feedback integral control.

Simulation results of the full motion model with the implemented linear state-feedback integral control show a system capable of reducing the swing motion of the suspended load. The swing oscillation caused by the relative motion of the platform is decreased in both frequency and amplitude by activating the designed state-space control system. Reference tracking of the tool-point position can be achieved with acceptable results while compensating for the swing-angles of the 3D pendulum.

The work conducted in this master thesis presents a method for developing a simulation model of a full motion system, where a control system is designed and implemented to act as an anti-swing system. Simulations results of the motion model yield an overall system capable of both robotic tool-point reference tracking and reducing the suspended load's swing-angles, with acceptable performance.

Bibliography

[1] J. Woodacre, R. Bauer, and R. Irani, "A review of vertical motion heave compensation systems," *Ocean Engineering*, vol. 104, pp. 140 – 154, 2015.

[2] M. B. Kjelland, *Offshore Wind Turbine Access Using Knuckle Boom Cranes*. phdthesis, University of Agder, 2016.

[3] S. S. Tørdal, P.-O. Løvsland, and G. Hovland, "Testing of wireless sensor performance in vessel-to-vessel motion compensation," in *Industrial Electronics Society, IECON 2016-42nd Annual Conference of the IEEE*, pp. 654–659, IEEE, 2016.

[4] S. S. Tørdal and G. Hovland, "Relative vessel motion tracking using sensor fusion, aruco markers, and mru sensors," 2017.

[5] S. S. T. rdal, W. Pawlus, and G. Hovland, "Real-time 6-dof vessel-to-vessel motion compensation using laser tracker," in *OCEANS 2017 - Aberdeen*, pp. 1–9, June 2017.

[6] Norwegian Motion Laboratory, "Equipment." `https://www.motion-lab.no/?page_id=9`. Accessed 03.03.2017.

[7] Norwegian Center for Offshore Wind and Energy, "Norcowe." `http://www.norcowe.no/`. Accessed 07.03.2017.

[8] S. S. Tørdal, G. Hovland, and I. Tyapin, "Efficient implementation of inverse kinematics on a 6-dof industrial robot using conformal geometric algebra," *Advances in Applied Clifford Algebras*, vol. 27, no. 3, pp. 2067–2082, 2017.

[9] Comau Robotics, "Nj 110-3.0." `http://www.comau.com/EN/our-competences/robotics/robot-team/nj-110-30`. Accessed 20.11.2017.

[10] Beckhoff, "Servomotor." `https://www.beckhoff.com/english.asp?drive_technology/am8532.htm`. Accessed 09.04.2017.

[11] Beckhoff, "Digital compact servo drives." `https://www.beckhoff.com/english.asp?drive_technology/ax51xx.htm`. Accessed 09.04.2017.

[12] Kongsberg Seatex, "Kongsberg seatex motion reference unit." `https://www.km.kongsberg.com/`. Accessed 22.03.2017.

[13] Qualisys, "Qualisys." `https://www.qualisys.com/`. Accessed 09.04.2017.

[14] Hexagon Manufacturing Intelligence, "Leica absolute tracker at960." `http://www.hexagonmi.com/en-IN/products/laser-tracker-systems/`. Accessed 09.04.2017.

[15] Beckhoff, "Beckhoff information system twincat 3." `http://www.beckhoff.com/english.asp?twincat/twincat-3-extended-automation-runtime.htm`. Accessed 10.04.2017.

[16] Beckhoff, "Beckhoff information system twincat 3 xar." `http://www.beckhoff.com/english.asp?twincat/twincat-3-extended-automation-runtime.htm?id=1893323218933308`. Accessed 10.04.2017.

[17] Beckhoff, "Beckhoff information system twincat 3 xae." http://www.beckhoff.com/english.asp?twincat/twincat-3-extended-automation-runtime.htm?id=1893323218933308. Accessed 10.04.2017.

[18] Beckhoff, "Beckhoff information system ads." https://infosys.beckhoff.com/english.php?content=../content/1033/tc3_c/9007200630561931.html&id=. Accessed 22.11.2017.

[19] Spong, *Robot Modeling and Control*. 2005.

[20] B. Siciliano, L. Sciavicco, L. Villani, and G. Oriolo, *Robotics: Modelling, Planning and Control (Advanced Textbooks in Control and Signal Processing)*. Springer, 2011.

[21] T. Gustafsson, *Modelling and Control of Offshore Crane Systems*. PhD thesis, Luleå University of Technology, 1993.

[22] O. Heng and S. S. Tørdal, "Calibration of the norwegian motion laboratory using conformal geometric algebra," in *CGI*, 2017.

[23] A. Emami-Naeini, J. D. Powell, and G. F. Franklin, *Feedback Control of Dynamic Systems, Global Edition*. Pearson Education Limited, 2014.

[24] M. E. Broucke, "Ece311 - dynamic systems and control linearization of nonlinear systems." http://www.control.utoronto.ca/~broucke/ece311s/Handouts/linearization.pdf. Accessed 09.04.2017.

[25] M. S. Fadali, "Linearization of nonlinear state equations." https://wolfweb.unr.edu/~fadali/ee471/Linearization.pdf. Accessed 20.04.2017.

[26] G. Welch and G. Bishop, "An introduction to the kalman filter." http://www.cs.unc.edu/~welch/media/pdf/kalman_intro.pdf, 1995. Accessed 10.04.2017.

[27] "A proposed spectral form for fully developed wind seas based on the similarity theory of s. a. kitaigorodskii," *Journal of Geophysical Research*, vol. 69, no. 24, pp. 5181–5190.

A Technical Specification

In this appendix the reader will be presented with extra information and specification of the equipment installed in the motion lab.

A.1 Stewart Platform

A.1.1 E-Motion 8000

Table A.1: E-Motion 8000 - Specification and Capacity [6]

Description	**Value**
Payload capacity $[kg]$	$m = 5500$
Maximum inertia $[kgm^2]$	$I_{xx} = 21274$
Maximum inertia $[kgm^2]$	$I_{yy} = 24193$
Maximum inertia $[kgm^2]$	$I_{zz} = 28197$
Surge $[m]$	$-1.110, +1.333$
Sway $[m]$	± 1.333
Heave $[m]$	$-0.955, +0.885$
Roll $[deg]$	± 26.10
Pitch $[deg]$	$-25.55, +33.40$
Yaw $[deg]$	± 31.10
Surge $[m/s]$	± 0.711
Sway $[m/s]$	± 0.711
Heave $[m/s]$	± 0.610
Roll $[deg/s]$	± 20.0
Pitch $[deg/s]$	± 20.0
Yaw $[deg/s]$	± 20.0
Surge $[m/s^2]$	± 0.60
Sway $[m/s^2]$	± 0.60
Heave $[m/s^2]$	± 0.80
Roll $[deg/s^2]$	± 100.0
Pitch $[deg/s^2]$	± 100.0
Yaw $[deg/s^2]$	± 100.0

A.1.2 E-Motion 1500

Table A.2: E-Motion 1500 - Specification and Capacity [6]

Description	**Value**
Payload capacity $[kg]$	$m = 1500$
Maximum inertia $[kgm^2]$	$I_{xx} = 2023$
Maximum inertia $[kgm^2]$	$I_{yy} = 3713$
Maximum inertia $[kgm^2]$	$I_{zz} = 3611$
Surge $[m]$	$-0.602, +0.716$
Sway $[m]$	± 0.603
Heave $[m]$	$-0.422, +0.407$
Roll $[deg]$	± 27.45
Pitch $[deg]$	$-24.35, +27.10$
Yaw $[deg]$	± 39.20
Surge $[m/s]$	± 0.8
Sway $[m/s]$	± 0.8
Heave $[m/s]$	± 0.6
Roll $[deg/s]$	± 40.0
Pitch $[deg/s]$	± 40.0
Yaw $[deg/s]$	± 40.0
Surge $[m/s^2]$	± 0.65
Sway $[m/s^2]$	± 0.60
Heave $[m/s^2]$	± 0.80
Roll $[deg/s^2]$	± 300.0
Pitch $[deg/s^2]$	± 300.0
Yaw $[deg/s^2]$	± 350.0

A.2 Comau Industrial Robot

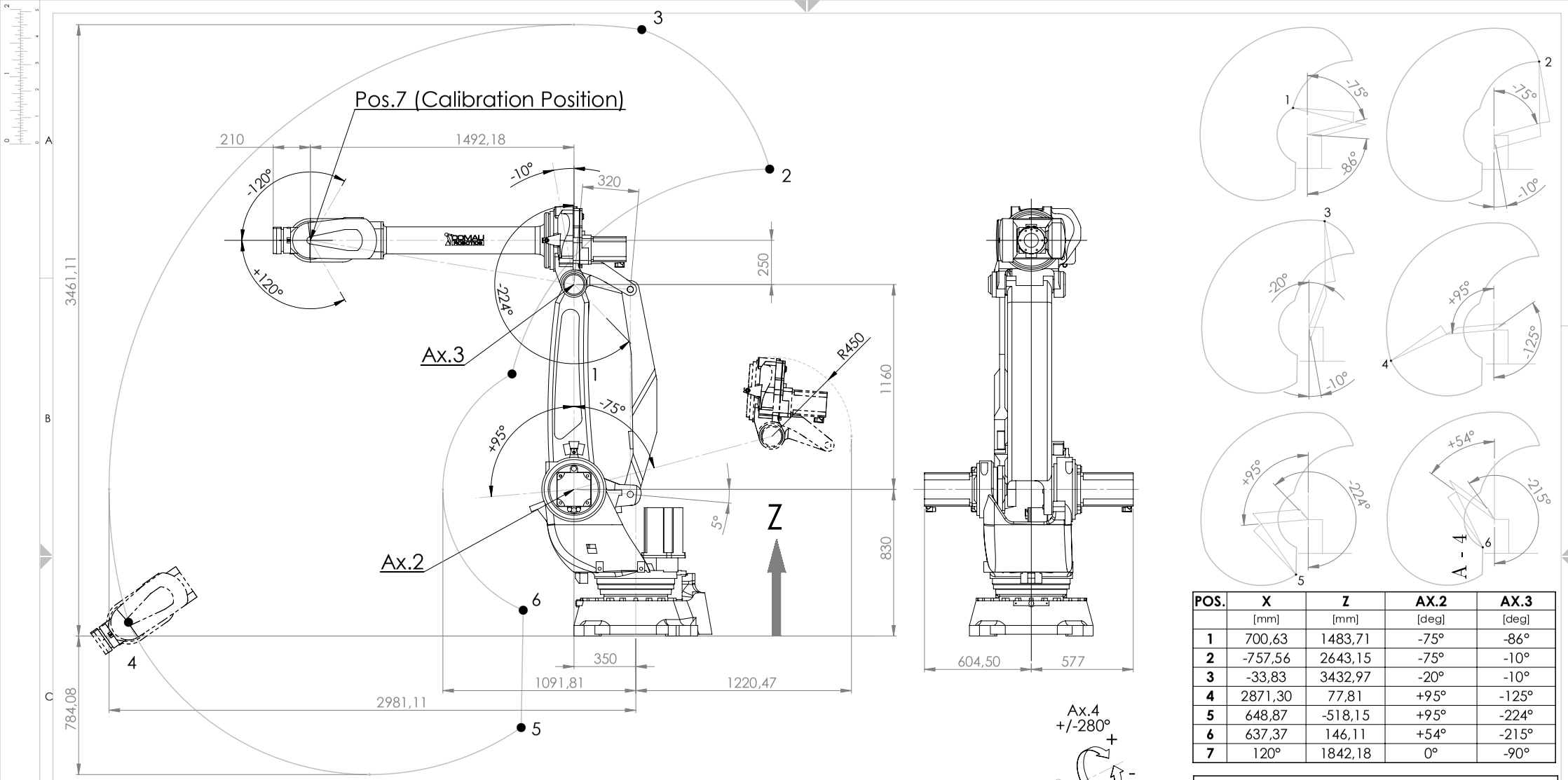
Table A.3: Comau Smart 5 NJ 110-3.0 - Technical Specification [9]

Description	**Value**
Number of axes	6
Maximum wrist payload $[kg]$	110
Additional load on forearm $[kg]$	50
Maximum horizontal reach $[mm]$	2980
Torque on axis 4 $[Nm]$	638
Torque on axis 5 $[Nm]$	638
Torque on axis 6 $[Nm]$	314
Stroke (Speed) on Axis 1 $[deg/s]$	± 180
Stroke (Speed) on Axis 2 $[deg/s]$	$-75, +95$
Stroke (Speed) on Axis 3 $[deg/s]$	$-10, -256$
Stroke (Speed) on Axis 4 $[deg/s]$	± 280
Stroke (Speed) on Axis 5 $[deg/s]$	± 120
Stroke (Speed) on Axis 6 $[deg/s]$	± 2700
Repeatability $[mm]$	0.07
Robot weight $[kg]$	1070
Tool coupling flange	ISO 9409 - A 125
Protection class	IP65 / IP67
Mounting position	Floor/Ceiling
Operating Areas A $[mm]$	3460
Operating Areas B $[mm]$	2980
Operating Areas C $[mm]$	2642
Operating Areas D $[mm]$	757
Operating Areas E $[mm]$	783

The full DH parameter table for the Comau Robot is listed in the table below. The constructed DH table is corrected for the disparities of the right-hand rotation and default home position between the provided Comau interface and Fig. 3.1.

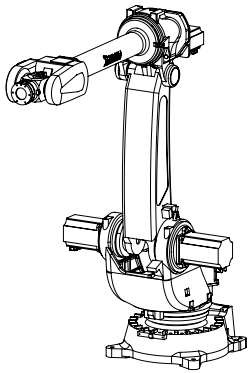
Table A.4: Denavit-Hartenberg Table for Comau Robot (6-DOF) without extension arm

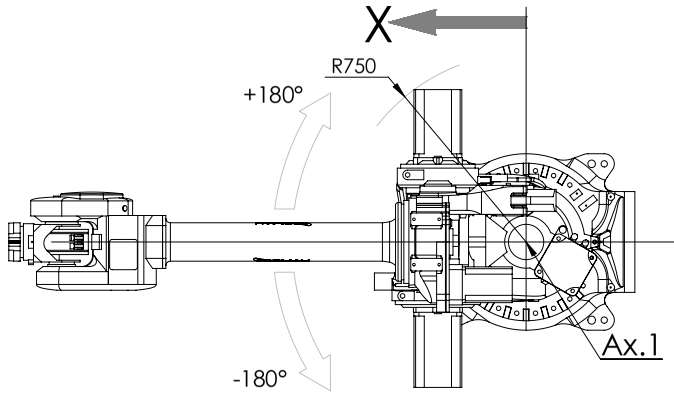
Link i	θ_i	d_i	a_i	α_i
1	$-\theta_1$	d_1	a_1	$\frac{\pi}{2}$
2	$\frac{\pi}{2} - \theta_2$	0	a_2	0
3	$\theta_3 + \frac{\pi}{2} + \theta_2$	0	a_3	$\frac{\pi}{2}$
4	$-\theta_4$	d_4	0	$-\frac{\pi}{2}$
5	$-\theta_5$	0	0	$\frac{\pi}{2}$
6	$\pi - \theta_6$	d_6	0	0

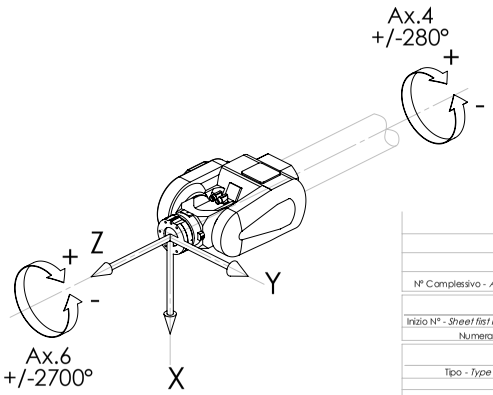


POS.	X	Z	AX.2	AX.3
	[mm]	[mm]	[deg]	[deg]
1	700,63	1483,71	-75°	-86°
2	-757,56	2643,15	-75°	-10°
3	-33,83	3432,97	-20°	-10°
4	2871,30	77,81	+95°	-125°
5	648,87	-518,15	+95°	-224°
6	637,37	146,11	+54°	-215°
7	120°	1842,18	0°	-90°

Joints in Calibration Position (Pos.7)					
Ax.1	**Ax.2**	**Ax.3**	**Ax.4**	**Ax.5**	**Ax.6**
0°	0°	-90°	0°	0°	0°







N° Complessivo - *Assembly Dwgs*	Quantita' per Complessivo *Q.ty Assembly*	N° Part. *Part. N°*

Inizio N° - *Sheet first N°*	Fine N° - *Sheet last N°*	Ultimo N° - *Last used*

Numerazione conduttori - *Wires numbering*

Tipo - *Type*	Codice - *Code*

Materiale - *Material*

Superficiale - *Surface*	Termico - *Heat*

Trattamento - *Treatment*

No.	Kg.	HRC — HB —
Modello - *Model*	Peso - *Weight*	Durezza - *Hardness*

Quote senza indicazioni di tolleranza : ISO 2768-mK

For dimensions with no tolerance : ISO 2768-mK

Tolleranze generali - *General tolerances*

N° - *No.*	Modifiche - *Revisions*	Data - *Date*	Visto - *Approv.*

Commessa - *Job*	18.10.10 Data - *Date*	1:15 Scala - *Scale*	A 2 Formato - *Size*
	1/1 Foglio - *Sheet*	Pernechele Disegn. - *Drawn*	Amparore Visto - *Checked*

Titolo — *Title*: SMART-5 NJ 110-3.0 AREA OPERATIVA

CR82345905

N° Disegno - *Drawing No.*

COMAU

B Maple Scripts

In this appendix, the Maple script used to derive the equation of motion for the 3-dimensional suspended load is presented.

```
> with(VectorCalculus)
  with(LinearAlgebra)
  with(VariationalCalculus)
  with(Physics)
```

(1)

[&x, `*`, `+`, `-`, `.`, <,>, <|>, About, AddCoordinates, ArcLength, BasisFormat, Binormal, Compatibility, ConvertVector, CrossProduct, Curl, Curvature, D, Del, DirectionalDiff, Divergence, DotProduct, Flux, GetCoordinateParameters, GetCoordinates, GetNames, GetPVDescription, GetRootPoint, GetSpace, Gradient, Hessian, IsPositionVector, IsRootedVector, IsVectorField, Jacobian, Laplacian, LineInt, MapToBasis, Nabla, Norm, Normalize, PathInt, PlotPositionVector, PlotVector, PositionVector, PrincipalNormal, RadiusOfCurvature, RootedVector, ScalarPotential, SetCoordinateParameters, SetCoordinates, SpaceCurve, SurfaceInt, TNBFrame, Tangent, TangentLine, TangentPlane, TangentVector, Torsion, Vector, VectorField, VectorPotential, VectorSpace, Wronskian, diff, eval, evalVF, int, limit, series]

[&x, Add, Adjoint, BackwardSubstitute, BandMatrix, Basis, BezoutMatrix, BidiagonalForm, BilinearForm, CARE, CharacteristicMatrix, CharacteristicPolynomial, Column, ColumnDimension, ColumnOperation, ColumnSpace, CompanionMatrix, CompressedSparseForm, ConditionNumber, ConstantMatrix, ConstantVector, Copy, CreatePermutation, CrossProduct, DARE, DeleteColumn, DeleteRow, Determinant, Diagonal, DiagonalMatrix, Dimension, Dimensions, DotProduct, EigenConditionNumbers, Eigenvalues, Eigenvectors, Equal, ForwardSubstitute, FrobeniusForm, FromCompressedSparseForm, FromSplitForm, GaussianElimination, GenerateEquations, GenerateMatrix, Generic, GetResultDataType, GetResultShape, GivensRotationMatrix, GramSchmidt, HankelMatrix, HermiteForm, HermitianTranspose, HessenbergForm, HilbertMatrix, HouseholderMatrix, IdentityMatrix, IntersectionBasis, IsDefinite, IsOrthogonal, IsSimilar, IsUnitary, JordanBlockMatrix, JordanForm, KroneckerProduct, LA_Main, LUDecomposition, LeastSquares, LinearSolve, LyapunovSolve, Map, Map2, MatrixAdd, MatrixExponential, MatrixFunction, MatrixInverse, MatrixMatrixMultiply, MatrixNorm, MatrixPower, MatrixScalarMultiply, MatrixVectorMultiply, MinimalPolynomial, Minor, Modular, Multiply, NoUserValue, Norm, Normalize, NullSpace, OuterProductMatrix, Permanent, Pivot, PopovForm, ProjectionMatrix, QRDecomposition, RandomMatrix, RandomVector, Rank, RationalCanonicalForm, ReducedRowEchelonForm, Row, RowDimension, RowOperation, RowSpace, ScalarMatrix, ScalarMultiply, ScalarVector, SchurForm, SingularValues, SmithForm, SplitForm, StronglyConnectedBlocks, SubMatrix, SubVector, SumBasis, SylvesterMatrix, SylvesterSolve, ToeplitzMatrix, Trace, Transpose, TridiagonalForm, UnitVector, VandermondeMatrix, VectorAdd, VectorAngle, VectorMatrixMultiply, VectorNorm, VectorScalarMultiply, ZeroMatrix, ZeroVector, Zip] [ConjugateEquation, Convex, EulerLagrange, Jacobi, Weierstrass] [`*`, `.`, Annihilation, AntiCommutator, Antisymmetrize, Assume, Bra, Bracket, Check, Christoffel, Coefficients, Commutator, Coordinates, Creation, D_, Dagger, Define, Dgamma, Einstein, Expand, ExteriorDerivative, FeynmanDiagrams, Fundiff, Geodesics, GrassmannParity, Gtaylor, Intc, Inverse, Ket, KillingVectors, KroneckerDelta, LeviCivita, Library, LieBracket, LieDerivative, Normal, Parameters, PerformOnAnticommutativeSystem, Projector, Psigma, Ricci, Riemann, Setup, Simplify, SpaceTimeVector, SubstituteTensor, SubstituteTensorIndices, SumOverRepeatedIndices, Symmetrize, TensorArray, Tetrads, ToFieldComponents, ToSuperfields, Trace, TransformCoordinates, Vectors, Weyl, `^`, dAlembertian, d_, diff, g_]

Suspended Load Kinematics

```
> "Kinematics";
  x_p := x_t(t) - L_w·sin(θ_x(t));
  y_p := y_t(t) + L_w·cos(θ_x(t))·sin(θ_y(t));
  z_p := z_t(t) - L_w·cos(θ_x(t))·cos(θ_y(t));
```

"Kinematics"

$$x_p := x_t(t) - L_w \sin\left(\theta_x(t)\right)$$
$$y_p := y_t(t) + L_w \cos\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right)$$
$$z_p := z_t(t) - L_w \cos\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \quad \textbf{(1.1)}$$

Lagrangian

> "Kinetic Energy";
$E_K := \frac{1}{2} m_p \cdot \left(diff(x_p, t)^2 + diff(y_p, t)^2 + diff(z_p, t)^2 \right);$
$simplify\left(expand\left(E_K \right), trig \right);$

"Kinetic Energy"

$$E_K := \frac{1}{2} m_p \left(\left(\frac{d}{dt} x_t(t) - L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \right)^2 + \left(\frac{d}{dt} y_t(t) \right.\right.$$
$$\left. - L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right)^2$$
$$\left. + \left(\frac{d}{dt} z_t(t) + L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \right)^2 \right)$$
$$\frac{1}{2} m_p \left(L_w^2 \cos\left(\theta_x(t)\right)^2 \left(\frac{d}{dt} \theta_y(t) \right)^2 - 2 \left(\frac{d}{dt} y_t(t) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) \right. \quad \textbf{(2.1)}$$
$$+ 2 \left(\frac{d}{dt} z_t(t) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right)$$
$$+ 2 \left(\frac{d}{dt} y_t(t) \right) L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right)$$
$$+ 2 \left(\frac{d}{dt} z_t(t) \right) L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) + L_w^2 \left(\frac{d}{dt} \theta_x(t) \right)^2$$
$$\left. - 2 \left(\frac{d}{dt} x_t(t) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) + \left(\frac{d}{dt} x_t(t) \right)^2 + \left(\frac{d}{dt} y_t(t) \right)^2 + \left(\frac{d}{dt} z_t(t) \right)^2 \right)$$

> "Potential Energy";
$E_P := m_p \cdot g \cdot z_p;$

"Potential Energy"

$$E_P := m_p g \left(z_t(t) - L_w \cos\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right) \quad \textbf{(2.2)}$$

> "Lagrangian";
$L := E_K - E_P;$

"Lagrangian"

$$L := \frac{1}{2} m_p \left(\left(\frac{d}{dt} x_t(t) - L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \right)^2 + \left(\frac{d}{dt} y_t(t) \right.\right. \quad \textbf{(2.3)}$$
$$\left. - L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right)^2$$
$$\left. + \left(\frac{d}{dt} z_t(t) + L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \right)^2 \right)$$
$$- m_p g \left(z_t(t) - L_w \cos\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right)$$

Euler Lagrange

> $L\_x := diff\left(diff\left(L, diff\left(\theta_x(t), t \right) \right), t \right) - diff\left(L, \theta_x(t) \right) = 0;$

$$L\_y := diff\left(diff\left(L, diff\left(\theta_y(t), t\right)\right), t\right) - diff\left(L, \theta_y(t)\right) = 0;$$

$$\begin{aligned}
L_x := \frac{1}{2} m_p \Bigg(& -2 \left(\frac{d^2}{dt^2} x_t(t) - L_w \left(\frac{d^2}{dt^2} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \right. \\
& \left. + L_w \left(\frac{d}{dt} \theta_x(t) \right)^2 \sin\left(\theta_x(t)\right) \right) L_w \cos\left(\theta_x(t)\right) + 2 \left(\frac{d}{dt} x_t(t) \right. \\
& \left. - L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) - 2 \left(\frac{d^2}{dt^2} y_t(t) \right. \\
& \left. - L_w \left(\frac{d^2}{dt^2} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) - L_w \left(\frac{d}{dt} \theta_x(t) \right)^2 \cos\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) \right. \\
& \left. - 2 L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right. \\
& \left. + L_w \cos\left(\theta_x(t)\right) \left(\frac{d^2}{dt^2} \theta_y(t) \right) \cos\left(\theta_y(t)\right) - L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right)^2 \sin\left(\theta_y(t)\right) \right) \\
& L_w \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) - 2 \left(\frac{d}{dt} y_t(t) - L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) \right. \\
& \left. + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) \\
& - 2 \left(\frac{d}{dt} y_t(t) - L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) \right. \\
& \left. + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right) L_w \sin\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \\
& + 2 \left(\frac{d^2}{dt^2} z_t(t) + L_w \left(\frac{d^2}{dt^2} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right. \\
& \left. + L_w \left(\frac{d}{dt} \theta_x(t) \right)^2 \cos\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right. \\
& \left. - 2 L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \right. \\
& \left. + L_w \cos\left(\theta_x(t)\right) \left(\frac{d^2}{dt^2} \theta_y(t) \right) \sin\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right)^2 \cos\left(\theta_y(t)\right) \right) \\
& L_w \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) + 2 \left(\frac{d}{dt} z_t(t) + L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right. \\
& \left. + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) - 2 \left(\frac{d}{dt} z_t(t) \right. \\
& \left. + L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \right) \\
& L_w \sin\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \sin\left(\theta_y(t)\right) \Bigg) - \frac{1}{2} m_p \left(2 \left(\frac{d}{dt} x_t(t) \right. \right. \\
& \left. - L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \right) L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) + 2 \left(\frac{d}{dt} y_t(t) \right. \\
& \left. - L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) + L_w \cos\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \right) \left(\right. \\
& -L_w \left(\frac{d}{dt} \theta_x(t) \right) \cos\left(\theta_x(t)\right) \sin\left(\theta_y(t)\right) - L_w \sin\left(\theta_x(t)\right) \left(\frac{d}{dt} \theta_y(t) \right) \cos\left(\theta_y(t)\right) \Big) \\
& + 2 \left(\frac{d}{dt} z_t(t) + L_w \left(\frac{d}{dt} \theta_x(t) \right) \sin\left(\theta_x(t)\right) \cos\left(\theta_y(t)\right) \right.
\end{aligned}$$

$$+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\Bigg)\left(L_w\left(\frac{d}{dt}\theta_x(t)\right)\cos\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)$$

$$-L_w\sin\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\Bigg)\Bigg)+m_p\,g\,L_w\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)=0$$

$$L\_y := \frac{1}{2}m_p\left(2\left(\frac{d^2}{dt^2}y_t(t)-L_w\left(\frac{d^2}{dt^2}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)\right.\right. \tag{2.1.1}$$

$$-L_w\left(\frac{d}{dt}\theta_x(t)\right)^2\cos\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)$$

$$-2L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)$$

$$+L_w\cos\left(\theta_x(t)\right)\left(\frac{d^2}{dt^2}\theta_y(t)\right)\cos\left(\theta_y(t)\right)-L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)^2\sin\left(\theta_y(t)\right)\Bigg)$$

$$L_w\cos\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)-2\left(\frac{d}{dt}y_t(t)-L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)\right.$$

$$\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)\right)L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)$$

$$-2\left(\frac{d}{dt}y_t(t)-L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)\right.$$

$$\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)\right)L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)$$

$$+2\left(\frac{d^2}{dt^2}z_t(t)+L_w\left(\frac{d^2}{dt^2}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)\right.$$

$$+L_w\left(\frac{d}{dt}\theta_x(t)\right)^2\cos\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)$$

$$-2L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)$$

$$\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d^2}{dt^2}\theta_y(t)\right)\sin\left(\theta_y(t)\right)+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)^2\cos\left(\theta_y(t)\right)\right)$$

$$L_w\cos\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)-2\left(\frac{d}{dt}z_t(t)+L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)\right.$$

$$\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\right)L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)+2\left(\frac{d}{dt}z_t(t)\right.$$

$$\left.+L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\right)$$

$$L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)\Bigg)-\frac{1}{2}m_p\left(2\left(\frac{d}{dt}y_t(t)\right.\right.$$

$$-L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)\Bigg)\Bigg($$

$$-L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)-L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\Bigg)$$

$$+2\left(\frac{d}{dt}z_t(t)+L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\cos\left(\theta_y(t)\right)\right.$$

$$\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\sin\left(\theta_y(t)\right)\right)\left(-L_w\left(\frac{d}{dt}\theta_x(t)\right)\sin\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)\right.$$

$$\left.\left.+L_w\cos\left(\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right)\cos\left(\theta_y(t)\right)\right)\right)+m_p\,g\,L_w\cos\left(\theta_x(t)\right)\sin\left(\theta_y(t)\right)=0$$

```
> θx" := solve(L_x, diff(θx(t), t, t));
  simplify(expand(θx"), 'symbolic');
```

$$\frac{d^2}{dx^2}\theta_x(x) := -\left(L_w\cos(\theta_x(t))\left(\frac{d}{dt}\theta_x(t)\right)^2\sin(\theta_x(t))\sin(\theta_y(t))^2 + L_w\cos(\theta_x(t))\left(\frac{d}{dt}\theta_x(t)\right)^2\sin(\theta_x(t))\cos(\theta_y(t))^2 + L_w\cos(\theta_x(t))\sin(\theta_x(t))\sin(\theta_y(t))^2\left(\frac{d}{dt}\theta_y(t)\right)^2 + L_w\cos(\theta_x(t))\sin(\theta_x(t))\left(\frac{d}{dt}\theta_y(t)\right)^2\cos(\theta_y(t))^2 - L_w\cos(\theta_x(t))\left(\frac{d}{dt}\theta_x(t)\right)^2\sin(\theta_x(t)) - \sin(\theta_x(t))\left(\frac{d^2}{dt^2}y_t(t)\right)\sin(\theta_y(t)) + \sin(\theta_x(t))\cos(\theta_y(t))\left(\frac{d^2}{dt^2}z_t(t)\right) + \sin(\theta_x(t))\cos(\theta_y(t))g - \left(\frac{d^2}{dt^2}x_t(t)\right)\cos(\theta_x(t))\right) \Big/ \left(L_w\left(\cos(\theta_y(t))^2\sin(\theta_x(t))^2 + \sin(\theta_x(t))^2\sin(\theta_y(t))^2 + \cos(\theta_x(t))^2\right)\right)$$

$$-\frac{1}{L_w}\left(L_w\cos(\theta_x(t))\sin(\theta_x(t))\left(\frac{d}{dt}\theta_y(t)\right)^2 - \sin(\theta_x(t))\left(\frac{d^2}{dt^2}y_t(t)\right)\sin(\theta_y(t)) + \sin(\theta_x(t))\cos(\theta_y(t))\left(\frac{d^2}{dt^2}z_t(t)\right) + \sin(\theta_x(t))\cos(\theta_y(t))g - \left(\frac{d^2}{dt^2}x_t(t)\right)\cos(\theta_x(t))\right) \quad \textbf{(2.1.2)}$$

```
> θy" := solve(L_y, diff(θy(t), t, t));
  simplify(expand(θy"), 'symbolic');
```

$$\frac{d^2}{dx^2}\theta_y(x) := \frac{1}{L_w\cos(\theta_x(t))\left(\cos(\theta_y(t))^2 + \sin(\theta_y(t))^2\right)}\left(2L_w\sin(\theta_x(t))\sin(\theta_y(t))^2\left(\frac{d}{dt}\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right) + 2L_w\sin(\theta_x(t))\cos(\theta_y(t))^2\left(\frac{d}{dt}\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right) - \sin(\theta_y(t))\left(\frac{d^2}{dt^2}z_t(t)\right) - \sin(\theta_y(t))g - \cos(\theta_y(t))\left(\frac{d^2}{dt^2}y_t(t)\right)\right)$$

$$\frac{1}{L_w\cos(\theta_x(t))}\left(2L_w\sin(\theta_x(t))\left(\frac{d}{dt}\theta_x(t)\right)\left(\frac{d}{dt}\theta_y(t)\right) - \sin(\theta_y(t))\left(\frac{d^2}{dt^2}z_t(t)\right) - \sin(\theta_y(t))g - \cos(\theta_y(t))\left(\frac{d^2}{dt^2}y_t(t)\right)\right) \quad \textbf{(2.1.3)}$$

```
>
```

C Matlab Scripts

C.1 Robot Model

This section will present the Matlab scripts related to the modelling of the Comau industrial robot.

C.1.1 Forward Kinematics - Symbolic Derivation

The following script is used to derive the symbolic governing equations for the forward kinematics.

```
%% Symbolic derivation of the Forward Kinematics
% This script is used to derive the governing equation for the
% forward kinematic of the Comau Robot

% Symbolic variables and Dimensions
syms q1 q2 q3
syms q1_t q2_t q3_t
syms q1_tt q2_tt q3_tt
syms d1 a1 a2 a3 L

% Variable Definitions
q = [q1; q2; q3];
q_t = [q1_t; q2_t; q3_t];
q_tt = [q1_tt; q2_tt; q3_tt];

% Constructing the DH-table
A1 = DH(-q1, d1, a1, sym(pi)/2);
A2 = DH(sym(pi)/2 - q2, 0, a2, 0);
A3 = DH(q3 + sym(pi)/2 + q2, 0, a3, sym(pi)/2);
A4 = DH(sym(pi), L, 0, 0);

% Transformation matrix
% (Robot base to tool-point {r} -> {t})
T04 = A1*A2*A3*A4;
T04 = simplify(expand(T04));  % Cleaner expression

% Rotation matrix
Rt = T04(1:3,1:3);

% Tool-point position
Pt = T04(1:3,4);

% Jacobian matrix
J = jacobian(Pt,q);

% Tool-point velocity
Pt_t = J*q_t;

% Jacobiandot
J_t = jacobian(J*q_t,q);

```

```
42 % Tool-point acceleration
43 Pt_tt = J_t*q_tt;
44
45 %% Functions
46
47 % Transformation Matrix  for Z-rotation
48 function Rz = RotZ(theta)
49     Rz = [cos(theta), -sin(theta), 0, 0;
50           sin(theta), cos(theta),  0, 0;
51           0         , 0         ,  1, 0;
52           0         , 0         ,  0, 1];
53 end
54
55 % Transformation Matrix  for X-rotation
56 function Rx = RotX(alpha)
57     Rx = [1, 0          , 0           , 0;
58           0, cos(alpha), -sin(alpha), 0;
59           0, sin(alpha), cos(alpha) , 0;
60           0, 0          , 0           , 1];
61 end
62
63 % Transformation Matrix for Z-translation
64 function Tz = TransZ(d)
65     Tz = [1, 0, 0, 0;
66           0, 1, 0, 0;
67           0, 0, 1, d;
68           0, 0, 0, 1];
69 end
70
71 % Transformation Matrix for X-translation
72 function Tx = TransX(a)
73     Tx = [1, 0, 0, a;
74           0, 1, 0, 0;
75           0, 0, 1, 0;
76           0, 0, 0, 1];
77 end
78
79 % Denavit-Hartenberg Table Convention
80 function A = DH(theta, d, a, alpha)
81     A = RotZ(theta)*TransZ(d)*TransX(a)*RotX(alpha);
82 end
```

C.1.2 Comau Robot System Block

The following script describes the Matlab system block for the Comau Robot, this includes forward kinematics configuration, inverse kinematics configuration and the motion configuration, where the latter includes disturbance from the Stewart platform.

```
classdef ComauRobotSystem < matlab.System

    % Calculates the forward or inverse kinematics of the Comau Robot
    % Calculates the motion of the Comau robot tool-point relative to the
    % neutral coordinate system of the Stewart Platform.
    % The Tool-point will be influenced from both the relative motion of
    % the Stewart Platform and the actuation of the robotic joints
    %
    % Forward Kinematics:
    %   Input:
    %       q     : Robot joint angular position (vector)
    %       q_t   : Robot joint angular velocity (vector)
    %       q_tt  : Robot joint angular acceleration (vector)
    %
    %   Output:
    %       Pt     : Tool-point position (vector)
    %       Pt_t   : Tool-point velocity (vector)
    %       Pt_tt  : Tool-point acceleration (vector)
    %
    % Inverse Kinematics:
    %   Input:
    %       Pt     : Tool-point position
    %       Pt_t   : Tool-point velocity
    %       Pt_tt  : Tool-point acceleration
    %   Output:
    %       q      : Robot joint angular position (vector)
    %       q_t    : Robot joint angular velocity (vector)
    %       q_tt   : Robot joint angular acceleration (vector)
    %
    % Stewart Motion:
    %   Input:
    %       q      : Robot joint angular position (vector)
    %       q_t    : Robot joint angular velocity (vector)
    %       q_tt   : Robot joint angular acceleration (vector)
    %       eta    : Stewart platform position
    %       v      : Stewart platform velocity
    %       v_t    : Stewart platform acceleration
    %   Output:
    %       Pt     : Tool-point position (relative to world coordinate)
    %       Pt_t   : Tool-point velocity (relative to world coordinate)
    %       Pt_tt  : Tool-point acceleration (relative to world coordinate)

    %% Properties

    % Properties that can be changed during execution
    properties

    end

    % Properties that can only be changed before execution
    properties(Nontunable)
        StringChoice = 'Forward Kinematics'; % Comau Robot Mode
    end
```

```
54
55      % Properties of the kinematic drop-down menu
56      properties(Hidden, Constant)
57          StringChoiceSet = matlab.system.StringSet({'Forward Kinematics', ...
58              'Inverse Kinematics', 'Stewart Motion'});
59      end
60
61      % Only accessible by class members
62      properties(Access = private)
63          % Static link lengths
64          a1 = 0.350;
65          a2 = 1.160;
66          a3 = 0.250;
67          d1 = 0.830;
68          d4 = 1.4922;
69          d6 = 0.210;
70          dt = 0.567;
71          L;
72
73          % Calibrated transformation matrices
74          Hgn;    % World to EM8000  {g} -> {n}
75          Hgq;    % World to EM1500  {g} -> {q}
76          Hnq;    % EM8000 to EM1500 {n} -> {q}
77          Hbr;    % EM8000 to Comau  {b} -> {r}
78      end
79
80      %% Methods for Simulink Interface
81
82      methods (Access = protected)
83
84          %% System block input-setup
85
86          % Set number of inputs to the Simulink System object
87          function num_inputs = getNumInputsImpl(obj)
88
89              % Number of inputs to the related mode
90              switch obj.StringChoice
91                  case {'Forward Kinematics'}
92                      num_inputs = 3;
93                  case {'Inverse Kinematics'}
94                      num_inputs = 3;
95                  case {'Stewart Motion'}
96                      num_inputs = 6;
97              end
98          end
99
100         % Set names of the inputs to the Simulink System object
101         function varargout = getInputNamesImpl(obj)
102             n = getNumInputsImpl(obj);  % Get number of inputs
103             varargout = cell(1,n);      % Define function return vector
104
105             % Set name of the inputs to the related mode
106             switch obj.StringChoice
107                 case {'Forward Kinematics'}
108                     varargout{1} = 'q';         % Set name of input port 1
109                     varargout{2} = 'q_t';       % Set name of input port 2
110                     varargout{3} = 'q_tt';      % Set name of input port 3
111                 case {'Inverse Kinematics'}
112                     varargout{1} = 'P';         % Set name of input port 1
113                     varargout{2} = 'P_t';       % Set name of input port 2
114                     varargout{3} = 'P_tt';      % Set name of input port 3
```

```
115                 case {'Stewart Motion'}
116                     varargout{1} = 'q';         % Set name of input port 1
117                     varargout{2} = 'q_t';       % Set name of input port 2
118                     varargout{3} = 'q_tt';      % Set name of input port 3
119 
120                     varargout{4} = 'eta';       % Set name of input port 4
121                     varargout{5} = 'v';         % Set name of input port 5
122                     varargout{6} = 'v_t';       % Set name of input port 6
123                 otherwise
124                     % Error catch
125                     msg = 'ERROR unknown kinematic type is chosen';
126                     error(msg);
127             end
128         end
129 
130         %% System block output-setup
131 
132         % Set number of output ports to the Simulink System object
133         function num_outputs = getNumOutputsImpl(~)
134             num_outputs = 3;
135         end
136 
137         % Set names of the output ports to the Simulink System object
138         function varargout = getOutputNamesImpl(obj)
139             n = getNumOutputsImpl(obj); % Get number of outputs
140             varargout = cell(1,n);      % Define function return vector
141 
142             % Set name of the outputs to the related mode
143             switch obj.StringChoice
144                 case {'Forward Kinematics'}
145                     varargout{1} = 'Pt';        % Set name of output port 1
146                     varargout{2} = 'Pt_t';      % Set name of output port 2
147                     varargout{3} = 'Pt_tt';     % Set name of output port 3
148                 case {'Inverse Kinematics'}
149                     varargout{1} = 'q';         % Set name of output port 1
150                     varargout{2} = 'q_t';       % Set name of output port 2
151                     varargout{3} = 'q_tt';      % Set name of output port 3
152                 case {'Stewart Motion'}
153                     varargout{1} = 'Pt';        % Set name of output port 1
154                     varargout{2} = 'Pt_t';      % Set name of output port 2
155                     varargout{3} = 'Pt_tt';     % Set name of output port 3
156                 otherwise
157                     % Error catch
158                     msg = 'ERROR unknown kinematic type is chosen';
159                     error(msg);
160             end
161         end
162 
163         %% System output calculation
164 
165         % Initialize System object states
166         % (One-time calculations)
167         function setupImpl(obj)
168             % Dimensions
169             obj.L = obj.d4 + obj.d6 + obj.dt;
170 
171             % Load calibration data
172             obj.load_calibration();
173         end
174 
175         % Reset System object states
```

```
176         function resetImpl(obj)
177
178         end
179
180         % System output and state update equations
181         function varargout = stepImpl(obj, varargin)
182
183             % Switch-statement to determine the mode
184             switch obj.StringChoice
185
186                 % Forward Kinematics
187                 case {'Forward Kinematics'}
188                     % Define inputs:
189                     q = varargin{1};
190                     q_t = varargin{2};
191                     q_tt = varargin{3};
192
193                     % Update states
194                     [Pt, Pt_t, Pt_tt] = obj.forward(q, q_t, q_tt);
195
196                     % Define outputs
197                     varargout{1} = Pt;
198                     varargout{2} = Pt_t;
199                     varargout{3} = Pt_tt;
200
201                 % Inverse Kinematics
202                 case {'Inverse Kinematics'}
203                     % Define inputs:
204                     Pt = varargin{1};
205                     Pt_t = varargin{2};
206                     Pt_tt = varargin{3};
207
208                     % Update states
209                     [q, q_t, q_tt] = obj.inverse(Pt, Pt_t, Pt_tt);
210
211                     % Define outputs
212                     varargout{1} = q;
213                     varargout{2} = q_t;
214                     varargout{3} = q_tt;
215
216                 % Motion
217                 case {'Stewart Motion'}
218                     % Define inputs:
219                     q = varargin{1};
220                     q_t = varargin{2};
221                     q_tt = varargin{3};
222
223                     eta = varargin{4};
224                     v   = varargin{5};
225                     v_t = varargin{6};
226
227                     % Calculate relative Tool-Point motion
228                     % {t}/{n} given in {n}
229                     [Pt, Pt_t, Pt_tt] = obj.motion(q, q_t, q_tt, ...
230                                                    eta, v, v_t);
231
232                     % Define outputs
233                     % (Tool-point relative to Stewart Platform)
234                     % {t}/{n} given in {n}
235                     varargout{1} = Pt;
236                     varargout{2} = Pt_t;
```

```
237                     varargout{3} = Pt_tt;
238             end
239         end
240     end
241
242     %% Methods for Matlab interface
243
244     methods
245
246         % Creates an constructor of the ComauRobot class
247         function obj = ComauRobotSystem()
248             obj.L = obj.d4 + obj.d6 + obj.dt;    % Horizontal length from
249                                                  % q3 to Tool-point
250         end
251
252         % Forward Kinematics
253         function [Pt, Pt_t, Pt_tt] = forward(obj, q, q_t, q_tt)
254             % Joint angular position
255             q1 = q(1);
256             q2 = q(2);
257             q3 = q(3);
258
259             % Tool-point position (derived by "symbolic_forward")
260             Pt = [
261                 -cos(q1)*(obj.L*sin(q3) + cos(q3)*obj.a3 - sin(q2)*obj.a2 - obj.a1);
262                 sin(q1)*(obj.L*sin(q3) + cos(q3)*obj.a3 - sin(q2)*obj.a2 - obj.a1);
263                 obj.L*cos(q3) - obj.a3*sin(q3) + cos(q2)*obj.a2 + obj.d1
264                 ];
265
266             % Tool-point velocity
267             J = obj.jacobian(q);
268             Pt_t = J*q_t;
269
270             % Tool-point acceleration
271             J_t = obj.jacobian_dot(q, q_t);
272             Pt_tt = J_t*q_t + J*q_tt;
273
274         end
275
276         % Inverse Kinematics
277         function [q, q_t, q_tt] = inverse(obj, Pt, Pt_t, Pt_tt)
278
279             Pt = [Pt; 1];
280             % Position components
281             xt = Pt(1);
282             yt = Pt(2);
283             zt = Pt(3);
284
285             % Joint 1 angle
286             q1 = -atan2(yt, xt);
287
288             % Using the method of a Two-link planar robot
289             % to find the last to joint angles
290
291             % Transformation matrix {r -> j2}
292             T1 = math3d.DH(-q1, obj.d1, obj.a1, pi/2);
293             Pq2t = math3d.InvH(T1)*Pt;      % TCP given in joint 2 {q2 -> t}
294             xq2t = Pq2t(1);
295             yq2t = Pq2t(2);
296
297             % Geometry calculationsyy
```

```
298             B = sqrt(obj.a3^2 + obj.L^2);
299             C = sqrt(xq2t^2 + yq2t^2);
300             D = (C^2 - obj.a2^2 - B^2) / (2*obj.a2*B);
301
302             alpha = atan2(-sqrt(1-D^2), D); % negative sign: elbow-up
303             phi = atan2(obj.a3, obj.L);
304
305             thetaA = atan2(yq2t, xq2t);
306             thetaB = atan2(B*sin(alpha),obj.a2 + B*cos(alpha));
307
308             % Joint 2 angle
309             q2 = pi/2 - (thetaA - thetaB);
310
311             % Joint 3 angle
312             q3 = alpha - phi - q2;
313
314             % Joint angular position
315             q = [q1; q2; q3];
316
317             % Joint angular velocity
318             J = obj.jacobian(q);
319             q_t = J\Pt_t;
320
321             % Joint angular acceleration
322             J_t = obj.jacobian_dot(q, q_t);
323             q_tt = J\(Pt_tt - J_t*q_t);
324
325         end
326
327         % Calculate the motion of the system
328         % relative to Stewart Platfrom neutral coordinate
329         function [Pt, Pt_t, Pt_tt] = motion(obj, q, q_t, q_tt, ...
330                                             eta, v, v_t)
331
332             % Body fixed velocity and acceleration skew matrices
333             Sw = math3d.Skew(v(4:6));
334             Sw_t = math3d.Skew(v_t(4:6));
335
336             % Ship/Stewart relative to static csys {n} -> {b}
337             Rnb = math3d.Rzyx(eta(4:6));
338             Rnb_t = Rnb*Sw;
339             Rnb_tt = Rnb*Sw*Sw + Rnb*Sw_t;
340
341             % Constant offset between stewart platform and robot base
342             % {b} -> {r}
343             r = obj.Hbr(1:3,4);
344             Rbr = obj.Hbr(1:3,1:3);
345
346             % Calculate Comau Robot forward kinematics
347             % {r} -> {t}
348             [P, P_t, P_tt] = obj.forward(q, q_t, q_tt);
349
350             % Tool-point position relative to Stewart platform
351             % {t}/{n} given in {n}
352             Pt = eta(1:3) + Rnb*(r + Rbr*P);
353
354             % Tool-point velocity relative to Stewart platform
355             % {t}/{n} given in {n}
356             Pt_t = v(1:3) + Rnb_t*(r + Rbr*P) + Rnb*(Rbr*P_t);
357
358             % Tool-point acceleration relative to Stewart platform
```

```
359             % {t}/{n} given in {n}
360             Pt_tt = v_t(1:3) + Rnb_tt*(r + Rbr*P) ...
361                    + 2*Rnb_t*(Rbr*P_t) + Rnb*(Rbr*P_tt);
362         end
363
364         % Jacobian
365         function J = jacobian(obj, q)
366             % Joint angular position
367             q1 = q(1);
368             q2 = q(2);
369             q3 = q(3);
370
371             % Jacobian (derived by "symbolic_forward")
372             J11 = sin(q1)*(obj.L*sin(q3) + cos(q3)*obj.a3 ...
373                 - sin(q2)*obj.a2 - obj.a1);
374             J12 = cos(q1)*cos(q2)*obj.a2;
375             J13 = -cos(q1)*(obj.L*cos(q3) - obj.a3*sin(q3));
376
377             J21 = cos(q1)*(obj.L*sin(q3) + cos(q3)*obj.a3 ...
378                 - sin(q2)*obj.a2 - obj.a1);
379             J22 = -sin(q1)*cos(q2)*obj.a2;
380             J23 = sin(q1)*(obj.L*cos(q3) - obj.a3*sin(q3));
381
382             J31 = 0;
383             J32 = -sin(q2)*obj.a2;
384             J33 = -obj.L*sin(q3) - cos(q3)*obj.a3;
385
386             J = [J11, J12, J13;
387                  J21, J22, J23;
388                  J31, J32, J33];
389         end
390
391         % Jacobian time differentiated
392         function J_t = jacobian_dot(obj, q, q_t)
393             % Joint angular position
394             q1 = q(1);
395             q2 = q(2);
396             q3 = q(3);
397
398             % Joint angular velocity
399             q1_t = q_t(1);
400             q2_t = q_t(2);
401             q3_t = q_t(3);
402
403             % Jacobian_dot (derived by "symbolic_forward")
404             J11_t = q1_t*cos(q1)*(obj.L*sin(q3) ...
405                   + cos(q3)*obj.a3 - sin(q2)*obj.a2 - obj.a1) ...
406                   - q2_t*sin(q1)*cos(q2)*obj.a2 ...
407                   + q3_t*sin(q1)*(obj.L*cos(q3) - obj.a3*sin(q3));
408             J12_t = -q1_t*sin(q1)*cos(q2)*obj.a2 ...
409                   - q2_t*cos(q1)*sin(q2)*obj.a2;
410             J13_t = q1_t*sin(q1)*(obj.L*cos(q3) - obj.a3*sin(q3)) ...
411                   - q3_t*cos(q1)*(-obj.L*sin(q3) - cos(q3)*obj.a3);
412
413             J21_t = -q1_t*sin(q1)*(obj.L*sin(q3) + cos(q3)*obj.a3 ...
414                    - sin(q2)*obj.a2 - obj.a1) ...
415                    - q2_t*cos(q1)*cos(q2)*obj.a2 ...
416                    + q3_t*cos(q1)*(obj.L*cos(q3) - obj.a3*sin(q3));
417             J22_t = -q1_t*cos(q1)*cos(q2)*obj.a2 ...
418                    + q2_t*sin(q1)*sin(q2)*obj.a2;
419             J23_t = q1_t*cos(q1)*(obj.L*cos(q3)- obj.a3*sin(q3)) ...
```

```
420                     + q3_t*sin(q1)*(-obj.L*sin(q3) - cos(q3)*obj.a3);
421
422             J31_t = 0;
423             J32_t = -q2_t*cos(q2)*obj.a2;
424             J33_t = q3_t*(-obj.L*cos(q3) + obj.a3*sin(q3));
425
426             J_t = [J11_t, J12_t, J13_t;
427                    J21_t, J22_t, J23_t;
428                    J31_t, J32_t, J33_t];
429         end
430
431         function load_calibration(obj)
432             % Load in calibration structure
433             motionlab = load('calib.mat');
434
435             % Transformation matrices found from calibration
436             obj.Hgn = motionlab.calib.WORLD_TO_EM8000.H;  % {g} -> {n}
437             obj.Hgq = motionlab.calib.WORLD_TO_EM1500.H;  % {g} -> {q}
438             obj.Hbr = motionlab.calib.EM8000_TO_COMAU.H;  % {b} -> {r}
439         end
440     end
441 end
```

C.2 Suspended Load System

This section introduces the Matlab scripts used to model the motion of the suspended load system, also known as the pendulum. This includes the scripts of pendulum dynamics and kinematics.

C.2.1 Pendulum Dynamics

```
function phi_tt = pendulumDynamics(Pt_tt, L, L_t, phi, phi_t)
    % Calculates the dynamics of the suspended load
    % Derives the differential equation for the euler angles
    %
    % Input:
    %   Pt_tt   : Tool-point acceleration
    %   phi     : Pendulum Angular position euler angles
    %   phit_t  : Pendulum Angular velocity euler angles
    %   L       : Wire length
    %   L_t     : Wire velocity
    %
    %   Output  :
    %   phit_t  : Pendulum Angular acceleration euler angles

    % Parameters
    g = 9.81;   % Gravity

    % Tool-point acceleration components
    xt_tt = Pt_tt(1);
    yt_tt = Pt_tt(2);
    zt_tt = Pt_tt(3);

    % Pendulum euler angles (angular position)
    phix = phi(1);
    phiy = phi(2);

    % Pendulum euler angles (angular velocity)
    phix_t = phi_t(1);
    phiy_t = phi_t(2);

     % Pendulum ODE of the euler angles
    % (found from euler-lagrange equation,
    %  derived by the use of Maple)
    phix_tt = (xt_tt*cos(phix) + yt_tt*sin(phix)*sin(phiy) ...
              - zt_tt*sin(phix)*cos(phiy) ...
              - g*sin(phix)*cos(phiy) ...
              - 2*L_t*phix_t ...
              - L*phiy_t^2*sin(phix)*cos(phix)) / L;

    phiy_tt = (- yt_tt*cos(phiy) - zt_tt*sin(phiy) ...
              - g*sin(phiy) + 2*L_t*phiy_t*cos(phix) ...
              + 2*L*phix_t*phiy_t*sin(phix)) ...
              / (L*cos(phix));

    phi_tt = [phix_tt; phiy_tt];
```

C.2.2 Pendulum Kinematics

```
function Pp = pendulumKinematics(Pt, phi, L)
    % Calculates the kinematics of the suspended load

    % Input:
    %   Pt_tt   : Tool-point position
    %   phi     : Pendulum Angular position euler angles
    %   L       : Wire length


    % Tool-point position
    xt = Pt(1);
    yt = Pt(2);
    zt = Pt(3);

    % Pendulum euler angles
    phix = phi(1);
    phiy = phi(2);

    % Pendulum position
    xp = xt - L*sin(phix);
    yp = yt + L*cos(phix)*sin(phiy);
    zp = zt - L*cos(phix)*cos(phiy);

    Pp = [xp; yp; zp];
end
```

C.3 Stewart Platform Motion

```
function [eta, v, v_t] = stewartMotion(eta, eta_t, eta_tt)
    % Calculate the relative Velocity and Acceleration component
    % of the Stewart platform
    %
    % Input:
    %   eta     : Orientation of {b} -> {n}
    %   eta_t   : Velocity of {b} -> {n}
    %   eta_tt  : Acceleration of {b} -> {n}
    %
    % Output:
    %   eta     : Orientation of {b} -> {n}
    %   v       : Body fixed velocity {n}/{b} given in {b}
    %   v_t      : Body fixed acceleration {n}/{b} given in {b}

% s : Angle Sequence
% ('zyx' is used for simulation)
% ('xyz' is used when motion data is obtained from MRU)
s = 'zyx';

% Velocity
J = vJacobian(eta, s);
v = J\eta_t;

% Acceleration
J_t = aJacobian(eta, eta_t, s);
v_t = J\(eta_tt - J_t*v);

end

function J = vJacobian(eta, s)
    phi = eta(4);
    theta = eta(5);
    psi = eta(6);

    if strcmp(s, 'xyz')
        T11 = cos(psi)/cos(theta);
        T12 = -sin(psi)/cos(theta);
        T13 = 0;

        T21 = sin(psi);
        T22 = cos(psi);
        T23 = 0;

        T31 = -(cos(psi)*sin(theta))/cos(theta);
        T32 = (sin(psi)*sin(theta))/cos(theta);
        T33 = 1;

        T = [T11,   T12,    T13;
             T21,   T22,    T23;
             T31,   T32,    T33];

        J = [eye(3), zeros(3);
             zeros(3), T];

    elseif strcmp(s, 'zyx')
        T11 = 1;
        T12 = (sin(phi)*sin(theta))/cos(theta);
```

```
58          T13 = (cos(phi)*sin(theta))/cos(theta);
59
60          T21 = 0;
61          T22 = cos(phi);
62          T23 = -sin(phi);
63
64          T31 = 0;
65          T32 = sin(phi)/cos(theta);
66          T33 = cos(phi)/cos(theta);
67
68          T = [T11,   T12,    T13;
69               T21,   T22,    T23;
70               T31,   T32,    T33];
71
72          J = [eye(3), zeros(3);
73               zeros(3), T];
74      end
75  end
76
77  function J_t = aJacobian(eta, eta_t, s)
78      phi = eta(4);
79      theta = eta(5);
80      psi = eta(6);
81
82      phi_t = eta_t(4);
83      theta_t = eta_t(5);
84      psi_t = eta_t(6);
85
86      if strcmp(s, 'xyz')
87
88          T11 = (cos(psi)*sin(theta)*theta_t ...
89                 - cos(theta)*sin(psi)*psi_t)/cos(theta)^2;
90          T12 = -(cos(theta)*cos(psi)*psi_t ...
91                + sin(theta)*sin(psi)*theta_t)/cos(theta)^2;
92          T13 = 0;
93
94          T21 = cos(psi)*psi_t;
95          T22 = -sin(psi)*psi_t;
96          T23 = 0;
97
98          T31 = -(cos(psi)*theta_t ...
99                - cos(theta)*sin(theta)*sin(psi)*psi_t)/cos(theta)^2;
100         T32 = (sin(psi)*theta_t ...
101              + cos(theta)*cos(psi)*sin(theta)*psi_t)/cos(theta)^2;
102         T33 = 0;
103
104         T = [T11,   T12,    T13;
105              T21,   T22,    T23;
106              T31,   T32,    T33];
107
108         J_t = [zeros(3), zeros(3);
109                zeros(3), T];
110
111     elseif strcmp(s, 'zyx')
112
113         T11 = 0;
114         T12 = (sin(phi)*theta_t ...
115             + cos(phi)*cos(theta)*sin(theta)*phi_t)/cos(theta)^2;
116         T13 = (cos(phi)*theta_t ...
117             - cos(theta)*sin(phi)*sin(theta)*phi_t)/cos(theta)^2;
118
```

```
119         T21 = 0;
120         T22 = -sin(phi)*phi_t;
121         T23 = -cos(phi)*phi_t;
122
123         T31 = 0;
124         T32 = (cos(phi)*cos(theta)*phi_t ...
125             + sin(phi)*sin(theta)*theta_t)/cos(theta)^2;
126         T33 = -(cos(theta)*sin(phi)*phi_t ...
127             - cos(phi)*sin(theta)*theta_t)/cos(theta)^2;
128         T = [T11,   T12,    T13;
129              T21,   T22,    T23;
130              T31,   T32,    T33];
131
132         J_t = [zeros(3), zeros(3);
133                zeros(3), T];
134     end
135 end
```

C.4 Control System Simple Pendulum

This section will include the Matlab scripts used for the development the controllers related the 2D pendulum system

C.4.1 Extended Kalman Filter Estimator

```
function f = f_simulink(x,u)
    % State-transition function for the Extended Kalman Filter
    % related to the 2D Pendulum system

    % Declaring system constants
    dt = 1e-3;  % Sample time [s]
    g = 9.81;   % Gravity [m/s^2]
    L = 2.0;    % Wire length [m]

    % State vector
    % x(1) : x
    % x(2) : x_t
    % x(3) : theta
    % x(4) : theta_t

 f = [x(1) + x(2)*dt;
      x(2) + u*dt;
      x(3) + x(4)*dt;
      x(4) + (-g*sin(x(3)) - cos(x(3))*u)*dt/L];
end
```

```
function h = h_simulink(x)
    % Measurement function for the Extended Kalman Filter
    % related to the 2D Pendulum system

    % State vector
    % x(1) : x
    % x(2) : x_t
    % x(3) : theta
    % x(4) : theta_t

    h = [x(1), x(3)];
end
```

C.4.2 Pendulum 2D Linear Control

```
%% Linear Control for the 2D Pendulum System
% This script contain the derivation of the linearized model,
% design of the Kalman Filter Estimator and the Extended Kalman Filter
% aswell as the controller schemes for the state-feedback pre-filter
% control, and the state-feedback integral control

clear all;
close all;
clc;

%% Initial Values

% Constants
dt = 1e-3;  % Time step
L = 2.0;    % Wire length
g = 9.81;   % Gravity [m/s^2]

% Initial values
x_init = 0;
x_t_init = 0;
theta_init = 0*pi/180;
theta_t_init = 0;

%% Symbolic

% Parameters
% syms g L 'real'

x = sym('x', [4,1], 'real');
u = sym('u', 'real');

% State vector
xt = x(1);
xt_t = x(2);
theta = x(3);
theta_t = x(4);

% Input vector
xt_tt = u;

% Non-linear function of system ODE
% x_t = f(x,u)
f = [xt_t;
     u;
     theta_t;
     (-g*sin(theta) - cos(theta)*xt_tt)/L];

% Non-linear function
% y = h(x,u)
h = [xt, theta];

% State-space
A_sym = jacobian(f, x);
B_sym = jacobian(f, u);
C_sym = jacobian(h, x);
D_sym = jacobian(h, u);

```

```
58 % Equilibrium point
59 u0 = 0;
60 x0 = [0; 0; 0; 0];
61
62 % Update the Matrices
63 % with the linearization around at equilibrium states
64 A_sym = subs(A_sym, [x,u],[x0,u]);
65
66 A_sym = subs(A_sym, [x,u],[x0,u]);
67 B_sym = subs(B_sym, [x,u],[x0,u]);
68 C_sym = subs(C_sym, [x,u],[x0,u]);
69 D_sym = subs(D_sym, [x,u],[x0,u]);
70
71
72 %% Simulink State-space
73
74 % Update the symoblic Matrices with constant values
75 % and create numeric Matrices
76 A = double(subs(A_sym));
77 B = double(subs(B_sym));
78 C = double(subs(C_sym));
79 D = double(subs(D_sym));
80
81 % C = C(1:2:3,:)    % Removing zero row entries
82 % State space system
83 G = ss(A,B,C,D);
84
85 Ob = obsv(G);
86 Cr = ctrb(G);
87 rank(Cr);
88 rank(Ob);
89
90 % Step response
91 % figure(1)
92 % step(G)
93
94 %% Estimator Design
95 % Covariance matrices
96 P = 1e-3;                                % Initial state covariance
97 Q = diag([0, 0.001^2, 0.01^2, 0.01^2]); % Process noise covariance
98 R = diag([0.001^2, 0.01^2]);            % Measurement noise covariance
99
100 %% State-Feedback gain
101 % Weighting factor
102 w = 10;
103
104 % Weight matrix for the states
105 Q_LQR = diag([10*w, 10*w, 1000*w, 10*w]);
106
107 % Weight matrix for the control
108 R_LQR = 1;
109
110 % LQR feedback gain
111 K = lqr(A,B,Q_LQR, R_LQR);
112
113 % New State-space model
114 Ar = A - B*K;
115
116 F = ss(Ar, B, C, D);
117 % figure(1)
118 % step(F)
```

```
119
120 %% Pre filter
121 Cm = [1, 0, 0, 0];               % Only direct output from input states
122 N = inv(Cm*inv(B*K - A)*B);     % Pre-filter constant
123
124 %% Integral control
125 Ai = [zeros(size(Cm,1)), -Cm;
126       zeros(size(A,1), size(Cm,1)), A];
127 Ai_old = [zeros(size(Cm,1)), Cm;
128       zeros(size(A,1), size(Cm,1)), A];
129 Bi = [zeros(size(Cm,1), size(B,2)); B];
130 Ci = [zeros(size(Cm,1)), Cm];
131 Di = zeros(size(Cm,1),size(Bi,2));
132
133 % Weight matrix for the states
134 % width additional error state(s)
135 Q_LQR = diag([100*w, 100*w, 10*w, 10000*w, 10*w]);
136
137 % Weight matrix for the control
138 R_LQR = 1;
139
140 % Calculating gains
141 KI = lqr(Ai, Bi, Q_LQR, R_LQR);
142 Ke = -KI(1);
143 Ko = KI(2:5);
```

C.5 Control System Suspended Load

This section introduces the Matlab scripts used to derive the functions related to the design of the Extended Kalman Filter and the different control systems used in the anti-swing control of the suspended load and robot system.

C.5.1 Extended Kalman Filter Estimator

```
% Preamble
clear all;
close all;
clc;

% This script computes the state-transition, measurement function,
% and the related Jacobian matrices for the Extended Kalman Filter
% used for the state estimation of the 3D pendulum system

%% Symbolic Derivation

% Parameters
syms dt 'real'                  % Time step
x = sym('x', [10,1], 'real');   % State vector
u = sym('u', [3,1], 'real');    % Input vector

% Constants
L = 2.0;    % Wire length [m]
L_t = 0;    % Wire length rate [m/s]
g = 9.81;   % Gravity [m/s^2]

% State vector
pt = x(1:3);
pt_t = x(4:6);
phi = x(7:8);
phi_t = x(9:10);

% Input vector
pt_tt = u(1:3);

% System ode
x_t = [pt_t;
       pt_tt;
       phi_t;
       pendulumDynamics(pt_tt, L, L_t, phi, phi_t)];

% State transition function
% x_t = f(x,u)
f = x + x_t*dt;
f = simplify(f);

% State transition Jacobian
F = jacobian(f,x);
F = simplify(F);

% Measurement function
h = [pt;
     zeros(3,1);
```

```
49      phi;
50      zeros(2,1)];
51  h = simplify(h);
52
53  % Measurement function Jacobian
54  H = jacobian(h,u);
55  H = simplify(H);
56
57  %% Make functions
58  % State transition
59  % matlabFunction(f, 'File', 'f.m', 'Vars', {x, u, dt});
60  % matlabFunction(F, 'File', 'fJacobian.m', 'Vars', {x, u, dt});
61
62  % Measurement
63  matlabFunction(h, 'File', 'h.m', 'Vars', {x});
64  matlabFunction(H, 'File', 'hJacobian.m', 'Vars', {x});
```

C.5.2 Pendulum 3D Linear Control

```
%% Linear Control for the 3D Pendulum System
% This scripts contains the derivation of the linarized model
% Selection of the Extended Kalman Filter estimator paramters
% aswell as different controller schemes

% Preamble
clear all;
close all;
clc;

%% Initial values

% Constants
dt = 1e-3;  % Time step
L = 2.0;    % Wire length
g = 9.81;   % Gravity [m/s^2]

% Initial values
% Position
x_init   = 2.6192*0;
y_init   = 0;
z_init   = 2.24*0;

phix_init = 0*pi/180;
phiy_init = 0*pi/180;

pt_init = [x_init, y_init, z_init];
phi_init = [phix_init, phiy_init];

% Velocity
x_t_init = 0;
y_t_init = 0;
z_t_init = 0;

phix_t_init = 0;
phiy_t_init = 0;

pt_t_init = [x_t_init, y_t_init, z_t_init];
phi_t_init = [phix_t_init, phiy_t_init];

init = [pt_init, pt_t_init, phi_init, phi_t_init];

%% Symbolic Linearization
% Parameters
x = sym('x', [10,1], 'real');   % State vector
u = sym('u', [3,1], 'real');    % Input vector

% Constants
L = 2.0;    % Wire length [m]
L_t = 0;    % Wire length rate [m/s]
g = 9.81;   % Gravity [m/s^2]

% State vector
pt = x(1:3);
pt_t = x(4:6);
phi = x(7:8);
phi_t = x(9:10);
```

```
58
59 % Input vector
60 pt_tt = u(1:3);
61
62 % Non-linear function of system ODE
63 % x_t = f(x,u)
64 f = [pt_t;
65      pt_tt;
66      phi_t;
67      pendulumDynamics(pt_tt, L, L_t, phi, phi_t)];
68
69 % Non-linear function
70 % y = h(x,u)
71 h = [pt; phi];
72
73 % State-space
74 A_sym = jacobian(f, x);
75 B_sym = jacobian(f, u);
76 C_sym = jacobian(h, x);
77 D_sym = jacobian(h, u);
78
79 % Equilibrium states
80 u0 = zeros(3,1);                                    % Input vector
81 x0 = [pt; zeros(3,1); zeros(2,1); zeros(2,1)];    % State vector
82
83 % Update the Matrices
84 % with the linearization around at equilibrium states
85 A_sym = subs(A_sym, [x,u],[x0,u0]);
86 B_sym = subs(B_sym, [x,u],[x0,u0]);
87 C_sym = subs(C_sym, [x,u],[x0,u0]);
88 D_sym = subs(D_sym, [x,u],[x0,u0]);
89
90 %% Simulink State-space
91
92 % Update the symoblic Matrices with constant values
93 % and create numeric Matrices
94 A = double(subs(A_sym));
95 B = double(subs(B_sym));
96 C = double(subs(C_sym));
97 D = double(subs(D_sym));
98
99 %% Extended Kalman Filter
100
101 Nx = length(x); % Number of states
102
103 % Initial state covariance
104 P_EKF = 1e-3;
105
106 % Process noise covariance
107 Q_EKF = eye(Nx)*0.001^2;
108 Q_EKF(1:3,1:3) = eye(3)*0.00001^2;  % Pt
109 Q_EKF(4:6,4:6) = eye(3)*0.001^2;    % Pt_t
110 Q_EKF(7:8,7:8) = eye(2)*0.001^2;    % phi
111 Q_EKF(9:10,9:10) = eye(2)*0.01^2;   % phi_t
112
113 % Process noise covariance
114 R_EKF = eye(Nx)*1^2;
115 R_EKF(1:3,1:3) = eye(3)*0.001^2;    % Pt
116 R_EKF(4:5,4:5) = eye(2)*0.01^2;     % phi
117
118 %% State-Feedback
```

```
% Weight
w = 10;

% Weight matrix for the states
Q_lqr = diag([ 100*w,   100*w,  100*w, ...  % Pt
                10*w,   10*w,   10*w, ...   % Pt_t
              1000*w, 1000*w, ...           % Phi
               100*w,  100*w]);             % Phi

% Weight matrix for the control
R_lqr = diag([1, 1, 1]);    % Size = columns of B

K = lqr(A,B,Q_lqr, R_lqr);

% New State-space model
Ar = A - B*K;
F = ss(Ar, B, C, D);

% Show step response
% figure(1)
% step(F)

%% Pre-Filter
% C_xyz = C(1:3,:);
C_xyz = C(1:3,:);               % Only direct output from input states
N = inv(C_xyz*inv(B*K - A)*B);  % Pre-filter constant

%% Integral Control
Ai = [zeros(size(C_xyz,1)), C_xyz;
      zeros(size(A,1), size(C_xyz,1)), A];
Bi = [zeros(size(C_xyz,1), size(B,2)); B];
Ci = [zeros(size(C_xyz,1)), C_xyz];
Di = zeros(size(C_xyz,1),size(Bi,2));

% Weight matrix for the states
Q_lqr_IC = diag([ 100*w,    100*w,  100*w, ...  % z (error states)
                  100*w,    100*w,  100*w, ...  % Pt
                   10*w,     10*w,   10*w, ...  % Pt_t
                 1000*w,   1000*w, ...          % Phi
                  100*w,    100*w]);            % Phi

% Weight matrix for the control
R_lqr_IC = diag([1, 1, 1]);    % Size = columns of B

KI = lqr(Ai, Bi, Q_lqr_IC, R_lqr_IC);
Ke = KI(:,1:3);
Ko = KI(:,4:end);

% New State-space model
Ar = Ai - Bi*KI;
Hi = ss(Ar, Bi, Ci, Di);

% Show step response
% figure(2)
% step(Hi)
```

C.6 Control System Motion System

This section includes the Matlab scripts used to derive the functions related to the design of the Extended Kalman Filter and the control system used for the non-linear full motion system.

C.6.1 Extended Kalman Filter Estimator

```
% This script computes the matlab functions for the state-transition,
% measurement function, and the related Jacobian matrices for the Extended
% Kalman Filter used for state estimation of the full motion system

%% Symbolic Derivation

% Parameters
syms dt 'real'                  % Time step
x = sym('x', [35,1], 'real');   % State Vector
u = sym('u', [4,1], 'real');    % Input Vector

% State vector
eta = x(1:6);
v = x(7:12);
v_t = x(13:18);
p = x(19:21);
p_t = x(22:24);
p_tt = x(25:27);
L = x(28);
L_t = x(29);
phi = x(30:31);
phi_t = x(32:33);
phi_tt = x(34:35);

% Input vector
p_tt = u(1:3);
L_ref = u(4);

% Tool-point motion relative to Stewart Platform neutral frame
% {t}/{n} given in {n}
[pt, pt_t, pt_tt] = toolPointMotion(eta, v, v_t, p, p_t, p_tt);

% System ODE
ode = [stewartJacobian(eta)*v;
       v_t;
       zeros(6,1);
       p_t;
       p_tt;
       zeros(3,1);
       L_t;
       winchMotion(L_ref, L, L_t);
       phi_t;
       pendulum_dynamics(pt_tt, phi, phi_t, L, L_t);
       zeros(2,1)];

% State transition function and Jacobian
f = x + ode*dt;
f = simplify(f);
```

```
F = jacobian(f,x);
F = simplify(F);

% Measurement function and Jacobian
h = [eta;
     v;
     zeros(6,1);
     p;
     p_t;
     zeros(3,1)
     L;
     0;
     phi;
     phi_t;
     zeros(2,1)];

h = simplify(h);
H = jacobian(h,x);
H = simplify(H);

% Make state transition function
matlabFunction(f, 'File', 'f.m', 'Vars', {x, u, dt});
matlabFunction(F, 'File', 'fJacobian.m', 'Vars', {x, u, dt});

% Make measurement function
matlabFunction(h, 'File', 'h.m', 'Vars', {x, dt});
matlabFunction(H, 'File', 'hJacobian.m', 'Vars', {x, dt});

model.f = f;
model.F = F;
model.h = h;
model.H = H;

%% Child Functions

function L_tt = winchMotion(L_ref, L, L_t)
    % Parameters
    Kdc = 1.0;
    omega = 4*2*pi;
    zeta = 0.7;

    % ODE
    L_tt = L_ref*Kdc*omega^2 - 2*zeta*omega*L_t - omega^2*L;
end

function [Pt, Pt_t, Pt_tt] = toolPointMotion(eta, v, v_t, P, P_t, P_tt)

    % Calibrated transformation matrix {b} -> {r}
    Hbr = [-0.4972, 0.8676, -0.0050,    -1.0820;
            0.8676, 0.4972,  0.0012,     1.5360;
            0.0036,-0.0037, -1.0000,    -1.0245;
                 0,      0,       0,     1.0000];

    % Body fixed velocity and acceleration skew matrices
    Sw = math3d.Skew(v(4:6));
    Sw_t = math3d.Skew(v_t(4:6));

    % Ship/Stewart relative to static csys {n} -> {b}
    Rnb = math3d.Rzyx(eta(4:6));
    Rnb_t = Rnb*Sw;
    Rnb_tt = Rnb*Sw*Sw + Rnb*Sw_t;
```

```
110
111     % Constant offset between stewart platform and robot base
112     % {b} -> {r}
113     r = Hbr(1:3,4);
114     Rbr = Hbr(1:3,1:3);
115
116     % Tool-point position relative to Stewart platform
117     % {t}/{n} given in {n}
118     Pt = eta(1:3) + Rnb*(r + Rbr*P);
119
120     % Tool-point velocity relative to Stewart platform
121     % {t}/{n} given in {n}
122     Pt_t = v(1:3) + Rnb_t*(r + Rbr*P) + Rnb*(Rbr*P_t);
123
124     % Tool-point acceleration relative to Stewart platform
125     % {t}/{n} given in {n}
126     Pt_tt = v_t(1:3) + Rnb_tt*(r + Rbr*P) ...
127            + 2*Rnb_t*(Rbr*P_t) + Rnb*(Rbr*P_tt);
128 end
129
130 function phi_tt = pendulum_dynamics(Pt_tt, phi, phi_t, L, L_t)
131
132     % Constant Parameters
133     g = 9.81;   % Gravity
134
135     % Tool-point acceleration components
136     xt_tt = Pt_tt(1);
137     yt_tt = Pt_tt(2);
138     zt_tt = Pt_tt(3);
139
140     % Pendulum euler angles (angular position)
141     phix = phi(1);
142     phiy = phi(2);
143
144     % Pendulum euler angles (angular velocity)
145     phix_t = phi_t(1);
146     phiy_t = phi_t(2);
147
148      % Pendulum ODE of the euler angles
149     % (found from euler-lagrange equation,
150     %  derived by the use of Maple)
151     phix_tt = (xt_tt*cos(phix) + yt_tt*sin(phix)*sin(phiy) ...
152               - zt_tt*sin(phix)*cos(phiy) ...
153               - g*sin(phix)*cos(phiy) ...
154               - 2*L_t*phix_t ...
155               - L*phiy_t^2*sin(phix)*cos(phix)) / L;
156
157     phiy_tt = (- yt_tt*cos(phiy) - zt_tt*sin(phiy) ...
158               - g*sin(phiy) + 2*L_t*phiy_t*cos(phix) ...
159               + 2*L*phix_t*phiy_t*sin(phix)) ...
160               / (L*cos(phix));
161
162     phi_tt = [phix_tt; phiy_tt];
163 end
164
165 % Spherical Pendulum kinematics
166 function Pp = pendulum_kinematics(Pt, phi, L)
167     % Tool-point position
168     % {r} -> {t}
169     xt = Pt(1);
170     yt = Pt(2);
```

```
171     zt = Pt(3);
172
173     % Pendulum euler angles
174     phix = phi(1);
175     phiy = phi(2);
176
177     % Pendulum position
178     % {r} -> {p}
179     xp = xt - L*sin(phix);
180     yp = yt + L*cos(phix)*sin(phiy);
181     zp = zt - L*cos(phix)*cos(phiy);
182
183     Pp = [xp; yp; zp];
184 end
185
186 % Calculate the jacobian of Stewart motion
187 function J = stewartJacobian(eta)
188     % Input:
189     %   eta : Motion driver
190     % Output:
191     %   J   : Jacobian Matrix
192
193     T = Tphi(eta(4:6), 'zyx');
194
195     J = [eye(3), zeros(3);
196          zeros(3), T];
197 end
198
199 % Transformation matrix
200 function T = Tphi(phi, type)
201     rx = phi(1);
202     ry = phi(2);
203     rz = phi(3);
204
205     cx = cos(rx);
206     sx = sin(rx);
207
208     cy = cos(ry);
209     sy = sin(ry);
210
211     cz = cos(rz);
212     sz = sin(rz);
213
214     T = eye(3);
215
216     if cy ~= 0.0
217         if strcmp(type, 'xyz')
218             T = [     cz,   -sz,  0;
219                    cy*sz, cy*cz,  0;
220                   -sy*cz, sy*sz, cy];
221
222         elseif strcmp(type, 'zyx')
223             T = [    cy, sx*sy, cx*sy;
224                       0, cx*cy,-sx*cy;
225                       0,    sx,    cx];
226         end
227         T = 1.0/cy*T;
228     end
229 end
```

C.6.2 Motion System Linear Control

```
% Preamble
clear all;
close all;
clc;

% This script is used to design the state-feedback integral controller for
% the full motion system
%% Initial Values

% Constant
g = 9.81;   % Gravity [m/s^2]
dt = 1e-3;  % Step time

% Robot joint angles
q1_init = 0;
q2_init = 0;
q3_init = -pi/2;

q_init = [q1_init, q2_init, q3_init];

% Stewart Platform
eta_init = zeros(6,1);
v_init = zeros(6,1);
v_t_init = zeros(6,1);

% Tool point {r} -> {t}
p_init = [2.6192; 0; 2.24];
p_t_init = zeros(3,1);
p_tt_init = zeros(3,1);

% Winch
L_init = 2.0;
L_t_init = 0;

% Pendulum
phix_init = 0*pi/180;
phiy_init = 0*pi/180;
phi_init = [phix_init; phiy_init];

phi_t_init = zeros(2,1);
phi_tt_init = zeros(2,1);

% Tool-Point {n}->{t}
Pt_init = [-2.395; 3.811; -3.255];

%% Extended Kalman Filter
Nx = size(x,1);    % Number of states
Nz = size(h,1);   % Number of measurements

% Process noise covariance
Q_EKF = eye(Nx)*0.001^2;
Q_EKF(13:18,13:18) = eye(6)*0.05^2; % V_t
Q_EKF(25:27,25:27) = eye(3)*0.05^2; % P_tt
Q_EKF(29,29) = 0.05^2;              % L_t
Q_EKF(32:33,32:33) = eye(2)*0.1^2;  % Phi_t
Q_EKF(32:33,32:33) = eye(2)*0.1^2;  % Phi_tt

```

```
58 % Measurement noise covariance
59 R_EKF = eye(Nz)*0.001^2;
60 R_EKF(1:6,1:6) = eye(6)*0.01^2;
61 R_EKF(7:12,7:12) = eye(6)*0.01^2;
62 R_EKF(30:31,30:31) = eye(2)*0.1^2;
63 R_EKF(31:32,31:32) = eye(2)*0.1^2;
64
65 % Initial state covariance
66 P_EKF = eye(Nx)*0.001^2;
67
68 % Initial states
69 EKF_init = [eta_init;
70             v_init;
71             v_t_init;
72             p_init;
73             p_t_init;
74             p_tt_init;
75             L_init;
76             L_t_init;
77             phi_init;
78             phi_t_init;
79             phi_tt_init];
80
81 %% Symbolic Derivation Controller
82
83 % State Vector
84 x = sym('x', [10,1], 'real');
85 % Input Vector
86 u = sym('u', [3,1], 'real');
87
88 % State vector
89 p = x(1:3);
90 p_t = x(4:6);
91 phi = x(7:8);
92 phi_t = x(9:10);
93
94 % Input vector
95 p_tt = u(1:3);
96
97 % Assigning Intial values to parameters of the winch
98 L = L_init;
99 L_t = L_t_init;
100
101 % Assuming no platform motion
102 eta0 = zeros(6,1);
103 v0 = zeros(6,1);
104 v_t0 = zeros(6,1);
105
106 % Tool-point motion relative to Stewart Platform neutral frame ({n} -> {t})
107 [Pt, Pt_t, Pt_tt] = toolPointMotion(eta0, v0, v_t0, p, p_t, p_tt);
108
109 f = [p_t;
110      p_tt;
111      phi_t;
112      pendulumDynamics(Pt_tt, L, L_t, phi, phi_t)];
113
114 % Non-linear function
115 % y = h(x,u)
116 h = [p;
117      p_t;
118      phi;
```

```
119      phi_t];
120
121 % State-space
122 A_sym = jacobian(f,x);
123 B_sym = jacobian(f,u);
124 C_sym = jacobian(h,x);
125 D_sym = jacobian(h,u);
126
127 % Equilibrium states
128
129 % Input vector
130 u0 = zeros(3,1); %; L_init];
131
132 % State vector
133 x0 = [[2.6919; 0; 2.24];    % P {r}->{t}
134       zeros(3,1);           % P_t
135       zeros(2,1);           % Phi
136       zeros(2,1)];          % Phi_t
137
138 % Update the Matrices
139 % with the linearization around at equilibrium states
140 A_sym = subs(A_sym, x, x0);
141 A_sym = subs(A_sym, u, u0);
142
143 B_sym = subs(B_sym, x, x0);
144 B_sym = subs(B_sym, u, u0);
145
146 C_sym = subs(C_sym, x, x0);
147 C_sym = subs(C_sym, u, u0);
148
149 D_sym = subs(D_sym, x, x0);
150 D_sym = subs(D_sym, u, u0);
151
152 %% State Space Model
153
154 % Update the symoblic Matrices with constant values
155 % and create numeric Matrices
156 A = double(subs(A_sym));
157 B = double(subs(B_sym));
158 C = double(subs(C_sym));
159 D = double(subs(D_sym));
160
161 % State space system
162 G = ss(A,B,C,D);
163
164 Ob = obsv(G);
165 Cr = ctrb(G);
166 rank(Cr);
167 rank(Ob);
168
169 %% State-Feedback
170
171 Nx = length(x); % Number of states
172
173 % Weighting factor
174 w = 10;
175
176 % Weight matrix for the states
177 Q_LQR = eye(Nx);
178
179 Q_LQR(1:3,1:3) = eye(3)*1000*w;     % P
```

```
Q_LQR(4:6,4:6) = eye(3)*10*w;       % P_t
Q_LQR(7:8,7:8) = eye(2)*1000*w; % Phi
Q_LQR(9:10,9:10) = eye(2)*100*w;    % Phi_t

% Weight matrix for the control
R_LQR = eye(size(B,2));    % Size = columns of B

% LQR feedback gain
K = lqr(A,B,Q_LQR, R_LQR);

% New State-space model
Ar = A - B*K;

F = ss(Ar, B, C, D);

% Show step response
% figure(1)
% step(F)

%% Integral Control

Cm = C(1:3,:); % Only direct output from input states

% New state-space system matrices with error states
Ai = [zeros(size(Cm,1)), Cm;
      zeros(size(A,1), size(Cm,1)), A];
Bi = [zeros(size(Cm,1), size(B,2)); B];
Ci = [zeros(size(Cm,1)), Cm];
Di = zeros(size(Cm,1),size(Bi,2));

% Weight matrix for the states
Q_lqr_IC = diag([ 100*w,    100*w,  100*w, ...  % z (error states)
                  100*w,    100*w,  100*w, ...  % Pt
                   10*w,     10*w,   10*w, ...  % Pt_t
                 1000*w,   1000*w, ...          % Phi
                  100*w,    100*w]);            % Phi

% New Intergrator state
Ni = size(Cm,1); % Number of states

% Weight matrix for the states
Q_LQR_IC = eye(Nx + Ni);

Q_LQR_IC(1:3,1:3) = eye(3)*1000*w;      % Integrator States
Q_LQR_IC(4:6,4:6) = eye(3)*100*w;       % P
Q_LQR_IC(7:9,7:9) = eye(3)*10*w;        % P_t
Q_LQR_IC(10:11,10:11) = eye(2)*1000*w;  % Phi
Q_LQR_IC(12:13,12:13) = eye(2)*100*w;   % Phi_t

% Weight matrix for the control
R_LQR_IC = eye(size(B,2));    % Size = columns of B

% % Calculating gains
KI = lqr(Ai, Bi, Q_LQR_IC, R_LQR_IC);
Ki = KI(:,1:Ni);
Ko = KI(:,Ni+1:end);
```

C.7 Animation and Plotting

Matlab scripts used for plotting and animation of the different systems models.

```
% Plot the coordinate system in 3D plots
% with default rgb-colors to the related axis.
function plot_coordinatesystem(H)
    % Inputs:
    %   H       :   Transformation-matrix (size = 4x4)

    % Pre-allocate
    axis = zeros(2,3);

    % Rotation and Translation
    d = H(1:3,4);   % Translation vector
    R = H(1:3,1:3); % Rotation matrix

    % Axis colors
    colors = {'r', 'g', 'b'};

    % Length of each axis in csys
    csys_len = 0.5;

    % Plot all 3 axis
    % (Looping through plotting one axis at a time)
    for i = 1:3
        axis(1,:) = d;                      % csys origo [x0 y0 z0]
        axis(2,:) = d + R(:,i)*csys_len;    % end point of axis [x y z]

        X = axis(:,1);  % Vector of the x-component
        Y = axis(:,2);  % Vector of the y-component
        Z = axis(:,3);  % Vector of the z-component

        % Plot one of the axis with the related color
        plot3(X, Y, Z, colors{i}, 'LineWidth', 1)
    end
end
```

```
% Plot a line in 3D plots
function plot_line(p1, p2)
    % Inputs:
    %   p1  :   Point 1 [x1; y1; z1]
    %   p2  :   Point 2 [x2; y2; z2]

    % Pre-allocate
    line = zeros(2,3);

    line(1,:) = p1;
    line(2,:) = p2;

    X = line(:,1);
    Y = line(:,2);
    Z = line(:,3);

    plot3(X, Y, Z, 'k', 'LineWidth', 1)
end
```

C.7.1 Robot Pose

```
function plot_robotpose(q, Hr)
    % Plot the pose of the Comau Robot
    % This includes links and local coordinate systems

    % Inputs:
    % Hr : Transformation matrix of the robot base [4x4]
    % q  : Robot joints [3x1]

    % Static link lengths
    a1 = 0.350;
    a2 = 1.160;
    a3 = 0.250;
    d1 = 0.830;
    d4 = 1.4922;
    d6 = 0.210;
    dt = 0.567;
    L = d4 + d6 + dt;

    % Joint angles
    q1 = q(1);
    q2 = q(2);
    q3 = q(3);

    % Robot DH Table
    T1 = math3d.DH(-q1, d1, a1, pi/2);
    T2 = math3d.DH(pi/2 - q2, 0, a2, 0);
    T3 = math3d.DH(q3 + pi/2 + q2, 0, a3, pi/2);
    T4 = math3d.DH(pi, L, 0, 0);

    % Relative transformation
    Hr1 = Hr*T1;    % Robot base to joint q1 {r -> q1}
    Hr2 = Hr1*T2;   % Robot base to joint q2 {r -> q2}
    Hr3 = Hr2*T3;   % Robot base to joint q3 {r -> q2}
    Hr4 = Hr3*T4;   % Robot base to tool-point {r -> t}

    % Plot coordinate systems
    plot_coordinatesystem(Hr);      % Base
    plot_coordinatesystem(Hr1);     % Joint q1
    plot_coordinatesystem(Hr2);     % Joint q2
    plot_coordinatesystem(Hr3);     % Joint q3
    plot_coordinatesystem(Hr4);     % Tool-Point

    % Plot links
    plot_line(Hr(1:3,4), Hr1(1:3,4));   % Base to joint q1
    plot_line(Hr1(1:3,4), Hr2(1:3,4));  % joint q1 to joint q2
    plot_line(Hr2(1:3,4), Hr3(1:3,4));  % joint q2 to joint q3
    plot_line(Hr3(1:3,4), Hr4(1:3,4));  % joint q3 to Tool-point
end
```

C.7.2 Suspended Load Pose

```
% Plot the position of the Pendulum
function plot_pendulum(Pt, Pp, Hr)
    % Inputs:
    % Pt    :   Tool-point position [3x]
    % Pp    :   Pendulum position [3x1]
    % Hr    :   Transformation matrix of the robot base [4x4]

    % Transforming the Position vector relative to world CSYS
    Pt = Hr*[Pt; 1];
    Pp = Hr*[Pp; 1];

    % Pendulum positions
    xp = Pp(1);
    yp = Pp(2);
    zp = Pp(3);

    % Plot wire
    plot_line(Pt(1:3), Pp(1:3));

    % Plot Load
    plot3(xp, yp, zp, 'ro', 'LineWidth', 5);
end
```

C.7.3 Stewart Pose

```
% Plot the pose of the stewart platform
% This includes the reference world coordinate
function Hgb = plot_stewartplatform(eta, Hg, Hgn)
    % Input:
    %   eta     : Stewart platform orientation
    %   Hg      : World coordinate system (Reference CSYS)
    %   Hgn     : Transformation matrix, World to EM8000 {g} -> {n}
    % Output:
    %   Hgb     : Transformation matrix, World to EM8000 {g} -> {b}

    % Motion from intial platform pose to motion
    % {n} -> {b}
    Hnb = eye(4);
    Hnb(1:3,1:3) = math3d.Rzyx(eta(4:6));   % Rotation matrix
    Hnb(1:3,4) = eta(1:3);                  % Translation vector

    % Relative transformation
    Hgb = Hgn*Hnb;  % World to EM8000 (motion) {g} -> {b}

    % Plot coordinate system
    plot_coordinatesystem(Hg);      % Reference
    plot_coordinatesystem(Hgn);     % EM8000 static
    plot_coordinatesystem(Hgb);     % EM8000 motion
end
```

C.7.4 Full System Motion

```
function stop = SimulinkMotionPlot(q, eta, Pt, Pp)
    % Creates a 3D-animation for the motion of the motion-lab system
    % Requires values for the Comau Robot joints and stewart platform
    % Input:
    %   q   : Robot joint angular position
    %   eta : Stewart platform orientation
    %   Pt  : Tool-point position
    %   phi : Pendulum Angular position euler angles
    % Output:
    %   stop :  Bool value to stop the anitmation

    % Persistent variables, to only be initialized once
    persistent h;           % Plot
    persistent motionlab;   % Calibration structure
    % Transformation matrices
    persistent Hgn;         % World to EM8000  {g} -> {n}
    persistent Hbr;         % EM8000 to Comau  {b} -> {r}
    persistent Hg;          % World coordinate system (Reference CSYS)

    stop = false;

    %% Initialization
    % One time intitialization of plotting settings
    if isempty(h)
        % Close previous all plots
        close all

        % Plotting setup
        h = figure('Name','Motion Lab Animation');

        xlabel('x-axis')
        ylabel('y-axis')
        zlabel('z-axis')
        xlim([-5, 3])
        ylim([-3, 5])
        zlim([0, 8])

        hold on;
        grid on;
        view(-30,20);

        % Load in calibration structure
        motionlab = load('calib.mat');

        % Transformation matrices found from calibration
        Hgn = motionlab.calib.WORLD_TO_EM8000.H;  % {g} -> {n}
        Hbr = motionlab.calib.EM8000_TO_COMAU.H;  % {b} -> {r}

        % Reference coordinate system (world coordinate)
        Hg = eye(4);
    end

    %% Drawing
    if ishandle(h)
        cla;

        % Plot the motion of the Stewart Platform
```

```
58          Hgb = plot_stewartplatformpose(eta, Hg, Hgn);
59
60          % Tranformation from World CSYS to Robot Base
61          Hgr = Hgb*Hbr;
62
63          % Plot Robot pose relative to input CSYS
64          plot_robotpose(q, Hgr);
65
66          % Plot Pendulum pose relative to input CSYS
67          plot_pendulum(Pt, Pp, Hgn);
68
69          drawnow;
70      end
71
72      %% Destructor
73      if ~ishandle(h)
74          stop = true;
75      end
76
77  end
```

C.8 Math3d Library

This section presents the functions related to the custom library *math3d*. This library is developed by Sondre Sanden Tørdal (supervisor of this thesis), where these functions are frequently inherited by the other scripts and functions used throughout this thesis.

```
function T = DH(theta, d, a, alpha)

T = [cos(theta),-sin(theta)*cos(alpha), sin(theta)*sin(alpha),a*cos(theta);
     sin(theta), cos(theta)*cos(alpha),-cos(theta)*sin(alpha),a*sin(theta);
     0         , sin(alpha)           , cos(alpha)           ,d           ;
     0         ,0                     ,0                     ,1          ];

end
```

```
function invH = InvH(H)
% Reverse homgoenous rigid motions
% More efficient than using inv(H)
%
% INPUTS:
% H         : Homogeneus transformation matrix 4x4
%
% OUT:
% invH      : Reversed rigid motion matrix 4x4


R = H(1:3,1:3);
d = H(1:3,4);

invH = eye(4,4);

invH(1:3,1:3) = R';
invH(1:3,4) = -R'*d;

invH(4,1:4) = [0,0,0,1];

end
```

```
function vSkew = Skew(v)

vSkew = [0,-v(3),v(2);
         v(3),0,-v(1);
         -v(2),v(1),0];
 end
```

```
function R = Rzyx(phi)

rx = phi(1);
ry = phi(2);
rz = phi(3);

R11 = cos(ry)*cos(rz);
R12 = cos(rz)*sin(rx)*sin(ry);
R13 = sin(rx)*sin(rz) + cos(rx)*cos(rz)*sin(ry);

R21 = cos(ry)*sin(rz);
R22 = cos(rx)*cos(rz) + sin(rx)*sin(ry)*sin(rz);
R23 = cos(rx)*sin(ry)*sin(rz) - cos(rz)*sin(rx);

R31 = -sin(ry);
R32 = cos(ry)*sin(rx);
R33 = cos(rx)*cos(ry);

R = [R11, R12, R13;
     R21, R22, R23;
     R31, R32, R33];
end
```

```
function T = Tphi(phi, type)

    rx = phi(1);
    ry = phi(2);
    rz = phi(3);

    cx = cos(rx);
    sx = sin(rx);

    cy = cos(ry);
    sy = sin(ry);

    cz = cos(rz);
    sz = sin(rz);

    T = eye(3);

    if cy ~= 0.0
        if strcmp(type, 'xyz')
            T = [     cz,   -sz,  0;
                   cy*sz, cy*cz,  0;
                  -sy*cz, sy*sz, cy];

        elseif strcmp(type, 'zyx')
            T = [    cy, sx*sy, cx*sy;
                      0, cx*cy,-sx*cy;
                      0,    sy,    cy];
        end

        T = 1.0/cy*T;
    end

end
```