UNIVERSITY OF AGDER

# Modelling, Simulation and Control of offshore crane

Develop a kinematic and dynamic crane model and study of several control designs

**Lisa Ann Williams**

**Supervisor**
Jing Zhou

*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
Department of Engineering
Faculty of Technology and Science

# Acknowledgements

This thesis is written and carried out as a part of the education in the master of Mechatronics at the University of Agder. It has been a very instructive and in interesting process, but also a challenging process. The opportunity to work with this project is highly appreciated.

First, special thanks to the supervisor, Jing Zhou, for her invaluable assistance and guidance. Through the work with the thesis, she has been available both in meetings and whenever assistance was needed, and quickly replied to questions on email. She has shown great interest to the thesis and introduced interesting solutions to control design.

Also, thank you very much to the class mate, Jakub Frazik, for his help with implementing a crane model in Simulink. He was also valuable with the work concerning the LQR, with his knowledge of how a LQR worked.

# Abstract

This master thesis is about Modelling, Simulation and Control of a MacGregor Active Heave Compensation (AHC) 250t crane operating on the supply vessel Gran Canyon. The crane model was developed mathematically using robot modeling theory including both kinematic and dynamic equations. This model was developed and simulated in Matlab and Simulink and further compared, where the two models showed equal results.

Control designs for an offshore crane can be developed in several ways, but in this thesis the control task only concerns position control of the crane and can be divided into two control tasks. The main goal is to determine the most suitable controller design for the two control tasks, which are as follows:

- Control of crane joints with the aim to get the joint angles to follow a desired joint angle, which is a sine wave with an amplitude of one, with as small error between desired and measured join angles as possible.
- Control of crane end-effector in vertical direction with the aim to get the end-effector position in z-direction to follow a desired end-effector position in z-direction with as small error between desired and measured position as possible. The desired position is a linear movement from 5.432m to 1m with a velocity of 0.1m/s. Then the end-effector should be kept steady at 1m.

The dynamic model of the crane was implemented in Simulink and various control designs were developed with the task of controlling the joint angles and the end-effector position in vertical direction, using the dynamic model as the plant. PID-, PI and PD-controller design and Linear-Quadratic Regulator (LQR) design were developed to perform control of joint angles and end-effector separately. Two inverse kinematics methods were developed with the aim of controlling the end-effector based on the kinematic equations. Using the inverse Jacobian for this purpose caused singularities, but using the transpose Jacobian instead made it possible to simulate the system.

Simulations showed that a PID-controller design had the best performance when controlling the joint angles, with a maximal error between desired joint angle and measure joint angle of $q_1error$ = $2.775 \cdot 10^{-3}[rad]$, $q_2error$ = $3.327 \cdot 10^{-3}[rad]$ and $q_3error$ = $6.268 \cdot 10^{-4}[rad]$. While a PD-controller design showed the best performance when controlling the end-effector position in vertical direction, with a maximal error between desired and measured position as $z_eerror$ = $2.826[mm]$.

# Contents

# List of Figures

# List of Tables

# Abbreviations and Descriptions

| | |
|---|---|
| **ACH** | Active Heave Compensation |
| **CAD** | Computer-aided Design |
| **D** | Derivative term |
| **DH** | Denavit-Hartenberg |
| **DOF** | Degree of Freedom |
| **End-effector** | A device at the end of a robot manipulator. |
| **I** | Integral term |
| **LQR** | Linear-Quadratic Regulator |
| **P** | Proportional term |
| **Trajectory** | A desired path for the end-effector. |

# Chapter 1

# Introduction

The main purposes of this master thesis are to develop a crane model based on kinematic and dynamic equations, develop several control designs for the crane, and further determine the most optimal control design for an offshore crane.

## 1.1 Background and Motivation

Offshore cranes play an important part in several marine operations. They are expected to perform a wide range of different tasks such as placing a payload safely on shore or on a vessel. Cranes that are placed on a vessel are affected by vessel motions caused by environmental forces such as wind, waves and current. Therefore, Crane dynamics and crane heave compensation need to be designed and analyzed carefully. Last year, a master thesis was provided by MacGregor with the aim of developing a platform to study coupled dynamics between the crane and a marine craft. This concerned a MacGregor AHC 250t crane operating on the supply vessel Gran Canyon, which is designed by the company Skipsteknisk AS in Ålesund. In the last few decades, ocean engineering-orientated automation has become a dominating research focus in several fields, and as a contribution to the mentioned master thesis the implementation of various control algorithms with the task to control the crane in a desired position needs to be examined.

## 1.2 Problem Specification and Limitations

This project is about Modelling, Simulation and Control of an offshore crane and concerns the MacGregor AHC 250t crane operating on the supply vessel Gran Canyon, which is designed by the company Skipsteknisk AS in Ålesund.

An important part of this projects is the literature study, where the purpose is to document already existing methods for modeling, simulation and control of offshore crane.

This project is delimited to concern only the crane, and not coupled dynamics between the crane, vessel, cable and payload. Therefore, the focus is to develop a mathematical model of the crane using existing research on kinematic and dynamic equations associated with offshore cranes, build a dynamic model of the crane and consider several control methods applied on the kinematic and dynamic crane model.

There will be developed control designs that will directly control the crane joints, where the control task is to get the joint angles to follow a desired joint angle with as small error between desired and measured joint angles as possible. Control designs that will directly control the end-effector position in vertical direction will be developed as well, with the control task to get the end-effector position to follow a desired end-effector position with as small error between desired and measure end-effector position as possible. The main goal of this work is to determine the most optimal controller design for both controller task.

## 1.3 Objectives

The tasks of this thesis can be divided into three sub tasks. These sub tasks are considered as the objectives of the project and are listed below.

- Literature study of modeling, simulation and control of offshore cranes.
- Modeling and simulation of crane kinematics and dynamics.
- Develop several control designs, simulate the systems and examine the results of each controller design.

## 1.4 Outline

Chapter 2 contains an overview of existing research and applications related to the modelling, simulation and control of offshore cranes. Chapter 3 introduces the theoretical expressions that are central for understand the modelling and control used in this thesis. Chapter 4 contains a description of the crane and how some simplifications are done to obtain a viable system for the modelling, simulation and control. Chapter 5 presents a mathematical model of the crane, which concerns both crane kinematics and dynamics. This chapter also presents the dynamic crane design in Simulink. Chapter 6 presents a Matlab and a Simulink model, based on the mathematical model, where the results from the two models will be compared. Chapter 7 introduces several control designs used to control the crane in a desired position. Chapter 8 presents the simulation results from all control designs and a subsequent discussion for both controller tasks. Finally, chapter 9 gives a conclusion of the work and recommendations of future work.

## 1.5 Introduction to Software

Softwares used for accomplishing this thesis are Matlab and Matlab/Simulink. Matlab is a mathematical software that uses a script language primarily for numerical computing, and Simulink is a simulation software where block diagrams are used for the modelling. Simulink can also be used to model a dynamic system as a Simscape or SimMechanics model, but this procedure is not used in this thesis.

A model of the crane kinematics and dynamics is developed in both in Matlab and Simulink, where the purpose is to compare results from the two models. A dynamic model is developed in Simulink. Finally, several control designs are developed in Simulink, where some of them are run in parallel with a Matlab script.

# Chapter 2

# Literature Review

This chapter contains an overview of existing research and applications related to modelling, simulation and control of offshore cranes. It explains existing methods for modelling the crane kinematics and dynamics, as well as the control system for offshore cranes, the software used for modelling and simulation of the crane, and some advantages and disadvantages regarding the methods and software.

## 2.1 Modelling of crane

According to Chu and Asæy [5], many approaches exist for deriving dynamic equations of a mechanical system. All methods have in common that they generate equivalent sets of equations, but the approach is depending on computation and analysis of different purposes.

### 2.1.1 Denavit-Hartenberg (DH) Convention

In mathematical robot modelling kinematic equations are used to describe the motion of the robot manipulator without taking torque and forces into consideration [2]. The kinematics for a robot with n number of links can be extremely complex, which is why simplifications are completely necessary in order to model the crane kinematics. The Denavit-Hartenberg convention ensures a systematic procedure to develop robot manipulator kinematics [2]. In the article "Integrated multi-domain system modelling and simulation for offshore crane operations" [4], Chu, Asøy, Ehlers and Zhang document how the assignment of reference frame and notations followed the Denavit-Hartenberg convention, and how the convention is used to solve the kinematic chains of a knuckle boom crane when the global reference frame 0 was attached to the base of the crane. Knowing the velocity of the end-effector of the crane, the velocity of the joints is calculated, and then the cylinder velocities can be described as a function of the joint velocities.

### 2.1.2 Lagrange's Approach

In the article "A MULTI-BODY DYNAMIC MODEL BASED ON BOND GRAPH FOR MARITIME HYDRAULIC CRANE OPERATIONS" [5], Chu and Æsøy introduce the Lagrange's approach, which is a method for deriving the dynamic equations. Here, the dynamics of a three degree of freedom (DOF) knuckle boom crane relies on the energy properties, where the difference between the kinetic and potential energy is essential. This method makes it possible to reduce the equations needed to describe the motion of the crane, as it uses generalized coordinates to describe the system instead of taking every single body with mass and inertia into consideration. This approach is appropriate for modelling with the use of bond graphs, and it avoids derivative causality problems when modelling nonlinear systems [5].

### 2.1.3 Bond Graph

Chu and Æsøy also show how a maritime crane lifting system comprised of a 3DOFs crane with three revolute joints, a winch, a segment of wire, and a pendulum load is modelled by developing a bond graph, which is a graphical representation of a physical dynamic system. This is a modelling technique that describes the energy structure of a physical system [5]. The models of the hydraulic actuators can easily be integrated due to the fact that the bond graph method is an energy based modelling approach. Bond graph theory provides an assembled description of physical systems athwart several energy fields, which makes interconnection of subsystems manageable [7].

## 2.2 Control of Offshore Cranes

Sun, Fang, Chen, Fu, and Lu explain in their paper [9] that over the past several decades the control problem for land-fixed cranes has been deeply studied, and that a lot of solutions for both linear and nonlinear control methods have been reported and many control strategies for land-fixed cranes have already been developed. As opposed to the land-fixed cranes, which are fixed and operated in the inertia, the ship-mounted cranes are influenced by external disturbance from sea waves, sea wind, ocean current etc. and are working in non-inertial frames. Due to those rough working conditions and the influence by various external disturbance the control problem for offshore cranes is highly demanding when the intention of the control is to place a payload precisely and smoothly to a desired location as fast as possible, without making the payload swing during the operation and avoid residual swing at the end.

Sun, Fang, Chen, Fu, and Lu looked at other literature of work denoted to the control of ship-mounted cranes and found some control strategies that were already developed and are listed below [9].

- A feed forward control with gain scheduling
- A control scheme consisting of a variable-gain observer and a variable-gain controller
- Predictionbased control
- Preview tracking control
- Nonlinear feedback control
- Sliding mode control (SMC)
- Composite control
- Linear matrix inequality-based control
- Delayed feedback control
- Active rate-based control
- Combination-based control
- External model based control

They found some advantages of using these control strategies, for instance that the control scheme with variable-gain observer and variable-gain controller is proved to be effective and that two nonlinear sliding mode controllers are developed and are proved to be effective and robust. Despite these advantages, most of these control strategies are based on simplified or reduced crane dynamics, which may cause system instability because it is difficult to avoid a swinging payload, due to the complicated working scenario of an offshore crane [9].

### 2.2.1 Nonlinear Stabilizing Control without Linearization or Approximation

As a reply to the concerns regarding the stability problems with the existing control methods, Sun, Fang, Chen, Fu, and Lu present a control design for offshore cranes based on the original nonlinear dynamics of the crane without any simplifications or approximations [9]. They present how the dynamics is transformed into a form that is more practical for such an approach, where the new control variables are defined. The Lyapunov control law and a closed-loop stability analysis are provided, and as far as they know this paper produces the first closed-loop control method which attain asymptotic results for an offshore crane, affected by ship roll and heave movement, without needing linearization and approximation of the original nonlinear dynamics. This method is compared to the existing methods using MATLAB/Simulink RTWT to verify that the performance of such a control method is better and more robust against external disturbances than the other control methods.

### 2.2.2 Dynamic Positioning Control

For a vessel actuated by two main thrusters, Rokseth, Skjong and Pedersen present the use of a Dynamic positioning control system (DP-control system), where the purpose is to provide reference signals for the thrusters so that the vessel can be controlled in three directions; surge, sway and yaw. This control system consists of position and angle set points, a second-order reference model to smooth the position and yaw angle set point into a reference signal, a position controller that calculates the desired thrust vector, a thrust allocation algorithm that uses the desired thrust vector to allocate the desired thrust force for each of the two thrusters and local thruster controllers that realize the thrust commands. A nonlinear observer was also needed to filter out high-frequency components of the measurements regarding position and angle [7].

## 2.3 Software used for Modelling and Simulation of Crane

Several software for modelling and simulation of crane already exist. The use of Computer-aided design (CAD) tools for modelling and simulation of offshore cranes has been improved considerably the last few decades, but models from CAD tools with detailed design are usually to complex to simulate, especially when the physical system is to complex or if a real-time performance of the simulation is required [3].

In the article "The Functional Mockup Interface - seen from an industrial perspective" [6], Bertsch, Ahle and Schulmeister introduce the White box modelling, which is about modelling the entire system with one consistent method using a modelling language that is suitable for different physical fields. An example of this approach is MODELICA. MODELICA is a multi-domain modelling language that allow us to model and simulate the combination of electrical, mechanical, thermodynamic, hydraulic, biological, control, event, real-time, etc using the same modelling language [8]. This type of modelling is equation based and has led to the development of softwarepackages, extensions and tool-boxes such as MATLAB/Simulink, SimulationX, 20-sim and OpenModelica [3].

Related to simulation of offshore cranes, Chu and Æsøy present modelling and simulation of a knuckle boom crane using a bond graph implemented in 20-sim [5]. 20-sim is a software tool that allows the implementation of bond graphs in the modelling and simulation [5].

Rokseth, Skjong and Pedersen also used bond graph implementation in sim-20 when simulating the crane system but emphasized that the bond graph model can be simulated in any software supporting script. This is since a bond graph easily provides the state equations [7]. A set of first order equations of motion from the bond graph can be extracted by hand and be used, for instance, in MATLAB for simulation. The bond graph can also be transformed into a block

diagram for a simulation in MATLAB Simulink. Despite of the possibilities they underscore that the advantages with software that supports bond graphs is avoidance of the tedious task of extracting the equations by hand or transforming the bond graph into a block diagram. However, a disadvantage of using Simulink is that it is hard to model interconnections, and it is harder to divide the system into subsystems.

# Chapter 3

# Robot Modelling Theory and Control Theory

The purpose of this chapter is to present and define the theoretical expressions that are central for understand the modelling and control used in this thesis. There are several types of offshore cranes, in various sizes and with different abilities, but common for most of them; they can be described using robot modelling theory. Therefore, this chapter includes a presentation of the kinematics and dynamics of a robot manipulator and the associated equations. Since control of offshore cranes is an essential part of this thesis, control methods such as LQR and PID, P-, PI and PD-controllers are also presented and explained. Two inverse kinematics control methods such as Jacobian inversion method and Jacobian transpose method are explained as well.

## 3.1 Kinematics

Kinematic equations are used to describe the relation between the individual joints of the robot manipulator and the position and orientation of the tool or end-effector. The main objective of the kinematics is to describe the motion of the robot manipulator without taking torque and forces into consideration. The problem of determining the kinematics can be divided into two parts, namely forward kinematics and inverse kinematics.

The information regarding kinematics of a robot manipulator is mainly collected from Spong's book [12], but some information is also inspired by [2] and [11].

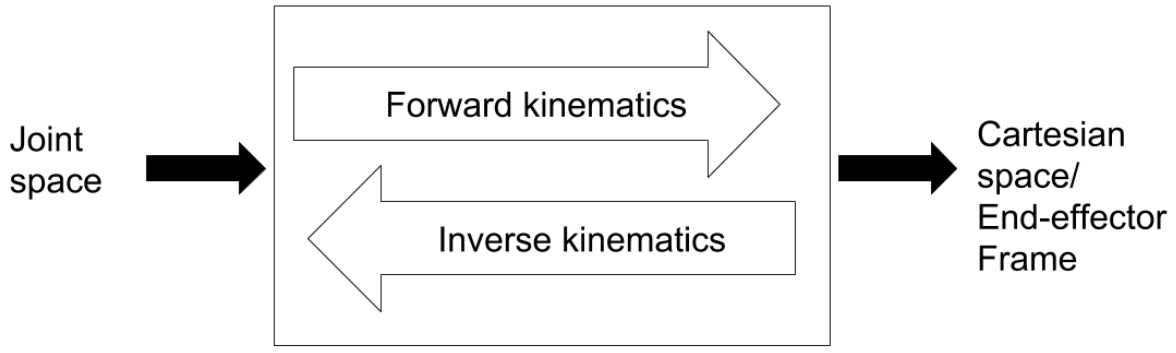



Figure 3.1: Forward and inverse kinematics

From Figure 3.1, we can distinguish between forward kinematics and inverse kinematics as follows:

- Forward kinematics is a method of calculating the motion of an end-effector from dimensions and states of the system on which it is mounted. From any given joint angle $q$ we can determine the resulting pose of the end-effector frame.
- Inverse kinematics is opposite of forward kinematics. It uses the kinematic equations to calculate the motion of the joints from the motion of the end-effector. From any desired pose of the end-effector frame, we can determine the required values for the joints $q$.

### 3.1.1 Robot Manipulator

Different kinds of robot manipulator are developed and can be classified by several criterion, such as the way the joints are actuated, their geometry or kinematic structure, their application area, or the method of control. In common, all robot manipulators can be described as a connection between a set of links and various joints. This connection makes it possible to move the end-effector of the robot manipulator to a desired point by modifying the joint angles.

### 3.1.2 Kinematic Chains

As mentioned in chapter 3.1.1, a robot manipulator is composed of a set of links connected together by various joints. These joints can be very simple, such as a revolute joint or a prismatic joint, or they can be complex, such as a ball and socket joint or a spherical wrist. A reovolute joint is like a hinge that allows a rotation about a single axis, and a prismatic joint allows a linear motion along a single axis, in other words an extension or retraction. A ball and socket have two degree of freedom and a spherical wrist has three degrees of freedom, so the benefits with the simple joints, compared to the complex joints, is that they only got a single degree of freedom, which is the angle of rotation in the case of a revolute joint and the amount of linear displacement in the case of a prismatic joint.

A robot manipulator with n joints consists of n+1 links, since each joint is connected by two links. The joints are numbered from 1 to $n$, and the links are numbered from 0 to $n$, starting from the base/ground. By this convention, joint $i$ connects link $i-1$ to link $i$. Joint 1 is the connection between link 0 and link 1, respectively the ground and the first link. Joint 2 is the connection between link 1 and 2, and so on. The location of joint $i$ is considered to be fixed with respect to link $i-1$. When the joint i is actuated link $i$ will move, but link 0, which is the first link, will therefore be fixed and does not move when the joints are actuated. All links in the configuration have one joint variable, which is denoted by $q_i$. For a revolute joint, $q_i$ is the angle of rotation, and for a prismatic joint, $q_i$ is the displacement of the joint:

$$q_i = \begin{cases} \theta_i & : \quad \text{Angle for a revolute joint} \\ d_i & : \quad \text{Displacement for a prismatic joint} \end{cases} \tag{3.1}$$

To perform a kinematic analysis, a coordinate frame is rigidly attached to each link. Link $i$ has coordinate frame $o_i x_i y_i z_i$, and the coordinate frame $o_0 x_0 y_0 z_0$ is attached to the base frame, which means that link 0 is the ground. Figure 3.2 illustrates how the coordinate frames are attached to the links of an elbow manipulator. The coordinate frames are placed according to the DH convention, which is further described in the next chapter.

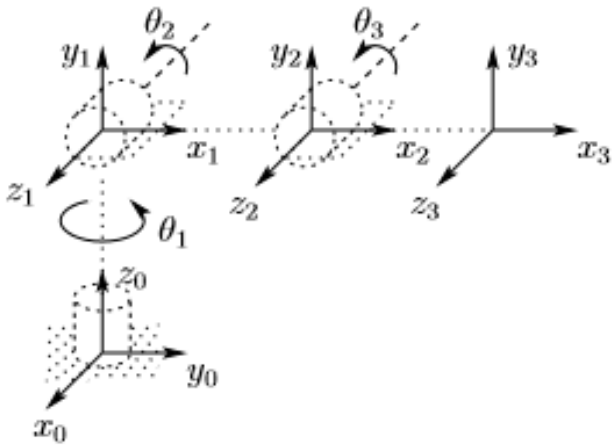



Figure 3.2: Coordinate frames of an elbow manipulator

### 3.1.3 DH Convention

When the kinematic analysis is carried out using an arbitrary frame attached to each link, it is helpful to be systematic in the choice of these frames. The DH convention is a commonly used convention for selecting frames. This convention simplifies the analysis significantly and provides a systematic procedure to develop robot manipulator kinematics. Table 3.1 shows the DH parameters for the elbow manipulator in Figure 3.2.

Table 3.1: DH parameters

| Link | $a_i$ | $\alpha_i$ | $d_i$ | $\theta_i$ |
|---|---|---|---|---|
| 1 | $l_1$ | $\alpha_1$ | $d_1$ | $\theta_1^*$ |
| 2 | $l_2$ | $\alpha_2$ | $d_2$ | $\theta_2^*$ |
| 3 | $l_3$ | $\alpha_3$ | $d_3$ | $\theta_3^*$ |

The parameters in this table is described as follows

- $a_i$ is the length of the links.
- $d_i$ is the link offset along the z-axis to the common normal. This is a variable joint parameter for a prismatic joint.
- $\alpha_i$ is the link twist. This is the angle between the common normal, from old z-axis to new z-axis, and the angle that aligns the $z_i$-axis with the joint-axis.
- $\theta_i$ is the joint angle and a variable joint parameter for a revolute joint. This is the rotational angle about the $z_i$-axis.

In this convention, we have four basic transformations representing translational and rotational movement relative to x and z, which are given by

$$Rot_z(\theta_i) = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i} & 0 & 0 \\ s_{\theta_i} & c_{\theta_i} & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

$$Rot_x(\alpha_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & c_{\alpha_i} & -s_{\alpha_i} & 0 \\ 0 & s_{\alpha_i} & c_{\alpha_i} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.3}$$

$$Trans_z(d_i) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.4}$$

$$Trans_x(a_i) = \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.5}$$

where each homogeneous transformations $A_i$ are represented as the product of the four basic transformations

$$\mathbf{A}_i^{i-1} = Rot_z(\theta_i)Trans_z(d_i)Trans_x(a_i)Rot_x(\alpha_i) \tag{3.6}$$

$$A_i = \begin{bmatrix} c_{\theta_i} & -s_{\theta_i}c_{\alpha_i} & s_{\theta_i}s_{\alpha_i} & a_i c_{\theta_i} \\ s_{\theta_i} & c_{\theta_i}c_{\alpha_i} & -c_{\theta_i}s_{\alpha_i} & a_i s_{\theta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.7}$$

### 3.1.4 Velocity and Acceleration Jacobian

For a joint $i$, the velocity $\dot{q}_i$ is related to the end-effector by

$$\begin{bmatrix} \mathbf{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \mathbf{J}\dot{\boldsymbol{q}} = \begin{bmatrix} \mathbf{J}_{vi} \\ \mathbf{J}_{\omega i} \end{bmatrix} \dot{\boldsymbol{q}}_i \tag{3.8}$$

where $\begin{bmatrix} \mathbf{J}_{vi} \\ \mathbf{J}_{\omega i} \end{bmatrix}$ is the geometric Jacobian for the joints.

The upper half of the Jacobian is the linear Jacobian, and is given as

$$\boldsymbol{J}_v = \begin{bmatrix} \boldsymbol{J}_{v_1} & \cdots & \boldsymbol{J}_{v_n} \end{bmatrix} \tag{3.9}$$

where the $i$-th column $\boldsymbol{J}_{v_i}$ is

$$\boldsymbol{J}_{v_i} = \boldsymbol{z}_{i-1} \times (\boldsymbol{r}_n - \boldsymbol{r}_{i-1}) \tag{3.10}$$

for a revolute joint and

$$\boldsymbol{J}_{v_i} = \boldsymbol{z}_{i-1} \tag{3.11}$$

for a prismatic joint.

The lower half of the Jacobian is the angular Jacobian, and is given by

$$\boldsymbol{J}_w = \begin{bmatrix} \boldsymbol{J}_{w_1} & \cdots & \boldsymbol{J}_{w_n} \end{bmatrix} \tag{3.12}$$

where the $i$-th column $\boldsymbol{J}_{w_i}$ is

$$\boldsymbol{J}_{w_i} = \boldsymbol{z}_{i-1} \tag{3.13}$$

for a revolute joint and

$$\boldsymbol{J}_{w_i} = 0 \tag{3.14}$$

for a prismatic joint.

$\boldsymbol{z}_{i-1}$ is the axis of rotation for joint $i$, $\boldsymbol{r}_n$ is the vector from the base frame to link $n$ and $\boldsymbol{r}_{i-1}$ is the vector from the base frame to link $i-1$.

When putting the upper and lower halves of the Jacobian together, the Jacobian for a n-link manipulator is of the form

$$\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{J}_1 & \boldsymbol{J}_2 & \cdots & \boldsymbol{J}_n \end{bmatrix} \tag{3.15}$$

where the i-th column $\boldsymbol{J}_i$ is

$$\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{J}_{v_i} \\ \boldsymbol{J}_{w_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_{i-1} \times (\boldsymbol{r}_n - \boldsymbol{r}_{i-1}) \\ \boldsymbol{z}_{i-1} \end{bmatrix} \tag{3.16}$$

for a revolute joint and

$$\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{J}_{v_i} \\ \boldsymbol{J}_{w_i} \end{bmatrix} = \begin{bmatrix} \boldsymbol{z}_{i-1} \\ 0 \end{bmatrix} \tag{3.17}$$

for a prismatic joint.

The total Jacobian for a n-link manipulator can then be written as

$$\boldsymbol{J}_i = \begin{bmatrix} \boldsymbol{J}_{v_i} \\ \boldsymbol{J}_{w_i} \end{bmatrix} = \begin{cases} \begin{bmatrix} \boldsymbol{z}_{i-1} \times (\boldsymbol{r}_n - \boldsymbol{r}_{i-1}) \\ \boldsymbol{z}_{i-1} \end{bmatrix} & \text{Revolute joint} \\ \begin{bmatrix} \boldsymbol{z}_{i-1} \\ 0 \end{bmatrix} & \text{Prismatic joint} \end{cases} \tag{3.18}$$

For a joint $i$, the acceleration of the end-effector is obtained by taking the time-derivative of the velocity equation, and is given by

$$\frac{d}{dt}\begin{bmatrix} \boldsymbol{v}_e \\ \boldsymbol{\omega}_e \end{bmatrix} = \begin{bmatrix} \dot{\boldsymbol{v}}_e \\ \dot{\boldsymbol{\omega}}_e \end{bmatrix} = \frac{d}{dt}(\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}}) = \frac{d}{dt}\boldsymbol{J}(\boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{J}(\boldsymbol{q})\ddot{\boldsymbol{q}} \tag{3.19}$$

## 3.2 Dynamics

While the kinematic equations describe the motion of the mechanical system without taking forces and moment into consideration, the dynamic equations specifically describe the relationship between force and motion.

All information regarding the dynamics of a robot manipulator is mainly obtained from Spong's book [12], but some information is also collected from [2] and [5].

### 3.2.1 Lagrange's Approach

Lagrange's approach is a method for deriving dynamic equations of a mechanical system. It relies on the energy properties of the system to compute the equations of motion, and because langrange's equations reduce the equations needed to describe the motion of the system by using generalized coordinates, it provides a fashionable formulation of the dynamics describing a mechanical system. The lagrange's approach is, as mentioned, an energy based approach of the system and derives the dynamics using kinetic and potential energy of the system. The lagrangian $\mathcal{L}$ of a mechanical system is described as the difference between kinetic and potential energy of the system, and is given by

$$\mathcal{L} = \mathcal{K} - \mathcal{P} \tag{3.20}$$

where $K$ and $P$ represent the total kinetic energy and the total potential energy, respectively.

Generally, for any type of mechanical system, the use of Lagrange's equations leads to a system of n coupled, second order nonlinear ordinary differential equations given by

$$\frac{d}{dt}\frac{\partial \mathcal{L}}{\partial \dot{q}_i} - \frac{\partial \mathcal{L}}{\partial q_i} = \tau_i \quad i = 1, ..., n \tag{3.21}$$

where $\tau_i$ is the force associated with link $i$.

The differential equations lay the foundations of the relation between the force applied to each joint and the joint positions, velocities and acceleration, and make it possible to derive the dynamic model using kinetic and potential energy of the system. It is also worth mentioning that the DH joint variables, described in chapter 3.1.3, provides a set of coordinates for a n-link rigid robot. The number of generalized coordinates are important to determine the order $n$ of the system, and are required to describe the evolution of the system.

### 3.2.2 Kinetic Energy

The total kinetic energy of a n-link manipulator can be written as the sum of contribution of kinetic energy relative to the motion of each link

$$\mathcal{K} = \sum_{i=1}^{n} \mathcal{K}_i \tag{3.22}$$

But first, let us look at kinetic energy of a rigid body, which can be written in the form

$$\mathcal{K}_i = \frac{1}{2} m_i \boldsymbol{v}_i^T \boldsymbol{v}_i + \frac{1}{2} \boldsymbol{\omega}_i^T \boldsymbol{\mathcal{I}}_i \boldsymbol{\omega}_i \tag{3.23}$$

where $m$ is the total mass of the object, $v$ is the linear velocity vector, $\omega$ is the angular velocity vector, and $\mathcal{I}$ is the Inertia tensor given by a $3 \times 3$ matrix. It is important to express the inertia tensor in the inertial frame to make it possible to compute the triple product $\boldsymbol{\omega_i^T \mathbf{I}_i \omega_i}$. This is done in terms of the orientation transformation between the body attached frame and the inertial frame, which leads to the inertia tensor given by

$$\boldsymbol{\mathcal{I}}_i = \boldsymbol{R}_i \boldsymbol{\mathcal{I}}_i \boldsymbol{R}_i^T \tag{3.24}$$

Both linear and angular velocities can be expressed by utilization of the Jacobian matrix and the derivative of the joint angles, and since the joint variables are the generalized coordinates, the linear and angular velocities can be written as

$$\boldsymbol{v}_i = \boldsymbol{J}_{v_i}(\boldsymbol{q})\dot{\boldsymbol{q}}, \quad \boldsymbol{\omega}_i = \boldsymbol{J}_{\omega_i}(\boldsymbol{q})\dot{\boldsymbol{q}} \tag{3.25}$$

By inserting Equation (3.23), (3.24) and (3.25) in Equation (3.22), the total kinetic energy for a n-link robot manipulator can be written as

$$\mathcal{K} = \frac{1}{2}\dot{\boldsymbol{q}}^T \sum_{i=1}^{n} [m_i \boldsymbol{J}_{v_i}(\boldsymbol{q})^T \boldsymbol{J}_{v_i}(\boldsymbol{q}) + \boldsymbol{J}_{\omega_i}(\boldsymbol{q})^T \boldsymbol{R}_i(\boldsymbol{q}) \boldsymbol{\mathcal{I}}_i \boldsymbol{R}_i(\boldsymbol{q})^T \boldsymbol{J}_{\omega_i}(\boldsymbol{q})] \dot{\boldsymbol{q}} \tag{3.26}$$

This equation can also be written as

$$\mathcal{K} = \frac{1}{2}\dot{\boldsymbol{q}} \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}} \tag{3.27}$$

where $\boldsymbol{M}(\boldsymbol{q})$ is the inertia matrix, and is given by

$$\boldsymbol{M}(\boldsymbol{q}) = m_i \boldsymbol{J}_{v_i}(\boldsymbol{q})^T \boldsymbol{J}_{v_i}(\boldsymbol{q}) + \boldsymbol{J}_{\omega_i}(\boldsymbol{q})^T \boldsymbol{R}_i(\boldsymbol{q}) \boldsymbol{\mathcal{I}}_i \boldsymbol{R}_i(\boldsymbol{q})^T \boldsymbol{J}_{\omega_i}(\boldsymbol{q}) \tag{3.28}$$

### 3.2.3 Potential Energy

As for the total kinetic energy, the total potential energy of a n-link manipulator can be written as the sum of contribution of potential energy relative to the motion of each link

$$\mathcal{P} = \sum_{i=1}^{n} \mathcal{P}_i \tag{3.29}$$

By assuming that the robot manipulator only consists of rigid links, the only force that causes potential energy is the gravity, therefore the potential energy for the $i$-th link can be computed as

$$\mathcal{P}_i = \boldsymbol{g}^T \boldsymbol{r}_{ci} m_i \tag{3.30}$$

where $g$ is a $3 \times 1$ gravity acceleration vector in the inertial frame and $r_c i$ is the vector of the center of mass of link $i$.

By inserting Equation (3.30) in Equation (3.29), the total potential energy for a n-link robot manipulator can be written as

$$\mathcal{P} = \sum_{i=1}^{n} \boldsymbol{g}^T \boldsymbol{r}_{ci} m_i \tag{3.31}$$

If the z-axis is defined as the vertical axis the gravity acceleration vector can be written as

$$\boldsymbol{g} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} \tag{3.32}$$

where $g = 9.81 m/s^2$.

### 3.2.4 Equations of Motion

Before the equations of motion can be derived, the Lagrange's equations (3.21) need to be spcialized. First, the kinetic energy can be written as a quadratic function of $\dot{\boldsymbol{q}}$ in the form

$$\mathcal{K} = \frac{1}{2} \sum_{i,j}^{n} M_{ij}(\boldsymbol{q}) \dot{q}_i \dot{q}_j = \frac{1}{2} \dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q}) \dot{\boldsymbol{q}} \tag{3.33}$$

and since the potential energy is independent of $\dot{\boldsymbol{q}}$, the potential energy can be written as

$$\mathcal{P} = \mathcal{P}(\boldsymbol{q}) \tag{3.34}$$

By inserting Equation (3.33) and (3.34) in Equation (3.20) the Lagrangian can be written as

$$\mathcal{L} = K - \mathcal{P} = \frac{1}{2}\sum_{i,j}^{n} M_{ij}(\boldsymbol{q})\dot{q}_i\dot{q}_j = \frac{1}{2}\dot{\boldsymbol{q}}^T \boldsymbol{M}(\boldsymbol{q})\dot{\boldsymbol{q}} - \mathcal{P}(\boldsymbol{q}) \tag{3.35}$$

Solving Equation (3.21) with respect to Equation (3.35) the Lagrange's equations can now be written as

$$\sum_j M_{kj}\ddot{q}_j + \sum_{i,j}\left\{\frac{\partial M_{kj}}{\partial q_i} - \frac{1}{2}\frac{\partial M_{ij}}{\partial q_k}\right\}\dot{q}_i\dot{q}_j - \frac{\partial \mathcal{P}}{\partial q_k} = \tau_k \tag{3.36}$$

Further, by interchanging the order of summation and use symmetry, we obtain the following Lagrange's equations

$$\sum_j M_{kj}(\boldsymbol{q})\ddot{q}_j + \sum_{i,j} C_{ijk}(\boldsymbol{q})\dot{q}_i\dot{q}_j + g_k(\boldsymbol{q}) = \tau_k, \quad k = 1, ..., n, \tag{3.37}$$

where $C_{ijk}$ is known as the Christoffel symbols and is given by

$$C_{ijk} = \frac{1}{2}\left\{\frac{\partial M_{kj}}{\partial q_i} + \frac{M_{ki}}{\partial q_j} - \frac{\partial M_{ij}}{\partial q_k}\right\} \tag{3.38}$$

and $g_k$ can be defined as

$$g_k = \frac{\partial \mathcal{P}}{\partial q_k}. \tag{3.39}$$

Finally, the equations of motion, also known as the manipulator equation, can be written in the matrix form

$$\boldsymbol{M}(\boldsymbol{q})\ddot{\boldsymbol{q}} + \boldsymbol{C}(\dot{\boldsymbol{q}}, \boldsymbol{q})\dot{\boldsymbol{q}} + \boldsymbol{g}(\boldsymbol{q}) = \boldsymbol{\tau}, \tag{3.40}$$

where $\boldsymbol{C}(\dot{\boldsymbol{q}}, \boldsymbol{q})$ is the Coriolis and Centripetal matrix, and the j,k-th matrix is defined as

$$C_{kj} = \sum_{i=1}^{n} C_{ijk}(\boldsymbol{q})q_i, \tag{3.41}$$

$\boldsymbol{M}(\boldsymbol{q})$ is the inertia matrix given in Equation (3.28) and $\boldsymbol{g}(\boldsymbol{q})$ is the gravity vector given in Equation (3.39).

## 3.3 Control Theory

To ensure a desired behaviour of a dynamic system, a controller is needed. This chapter includes a description of different controller methods. First, state-of-art feedback controllers such as a PID-, P-, PI- and PD controllers are explained. The structure of the controllers is shown with block diagram and equations, and how the controller parameters affect the system is described. In addition, Ziegler-Nichols open- and closed method for tuning a system is accounted for. Then, the LQR, which is based on state-space equations, is explained. This controller is also shown with block diagrams and equations describing the controller design. Finally, two inverse kinematics control methods such as Jacobian inversion method and Jacobian transpose method are explained, also with block diagram and equations.

All information regarding the PID-, P-,PI- and PD controller is obtained from [13], [14] and [15]. All information regarding the LQR are obtained from [16], [17], [18], [19], and [20]. All information regarding Jacobian transpose method are obtained from [21].

### 3.3.1 PID-controller

A PID-controller is a controller consisting of a proportional gain, an integral gain and a derivative gain, as shown in Figure 3.3.

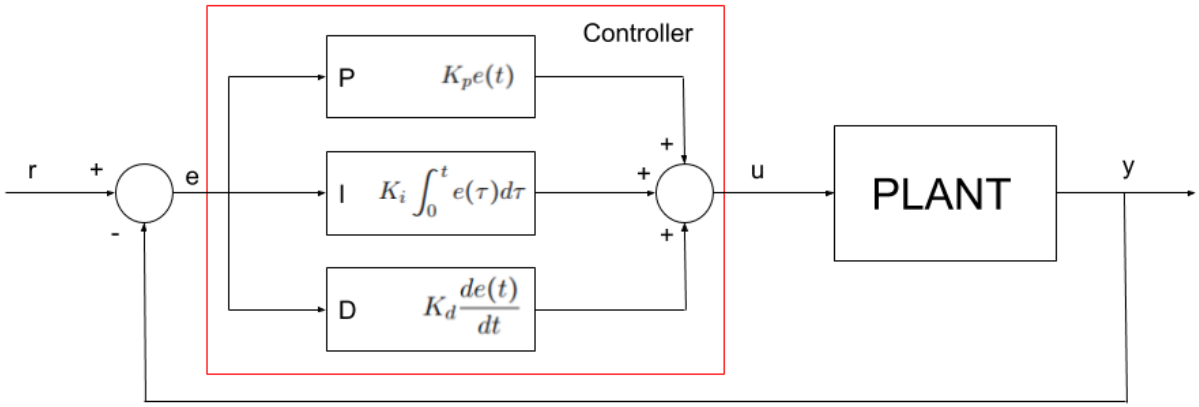



Figure 3.3: PID-controller

The output of a PID-controller, which is equal to the control input to the plant, is calculated in the time domain from the feedback error as follows

$$PID = K_p e(t) + K_i \int_0^t e(\tau)d\tau + K_d \frac{de(t)}{dt} = K_p(e(t) + \frac{1}{T_i}\int_0^t e(\tau)d\tau + T_d \frac{de(t)}{dt}) \tag{3.42}$$

where

- $K_p$ is the proportional gain
- $K_i$ is the integral gain
- $K_d$ is the derivative gain
- $T_i$ is the integral time constant
- $T_d$ is the derivative time constant

In Laplace domain the output, and transfer function, of the controller is given by

$$PID(s) = K_p + \frac{K_i}{s} + K_d s = \frac{K_d s^2 + K_p s + K_i}{s} = K_p(1 + \frac{1}{T_i s} + T_d s) \tag{3.43}$$

Changing the controller parameters will have different effects on the system response. For instance, increasing the proportional gain ($K_p$) will reduce, but not eliminate, the steady-state error, adding an integral term to the controller ($K_i$) also tends to help reduce steady-state error, and adding a derivative term to the controller ($K_d$) adds the ability of the controller to anticipate error. In addition, there are several other general effects of each controller parameters ($K_p$, $K_d$, $K_i$) for a closed-loop system, which are summarized in Table 3.2.

Table 3.2: Effects of controller parameters for a PID

| CL RESPONSE | RISE TIME | OVERSHOOT | SETTLING TIME | S-S ERROR |
|---|---|---|---|---|
| $K_p$ | Decrease | Increase | Small Change | Decrease |
| $K_i$ | Decrease | Increase | Increase | Decrease |
| $K_d$ | Small Change | Decrease | Decrease | No Change |

The use of a PID-controller ensures optimum control dynamics with zero steady state error, fast response, a higher stability and no oscillation.

### 3.3.2 P-controller

A P-controller is a controller consisting of only a proportional gain, as shown in Figure 3.4.

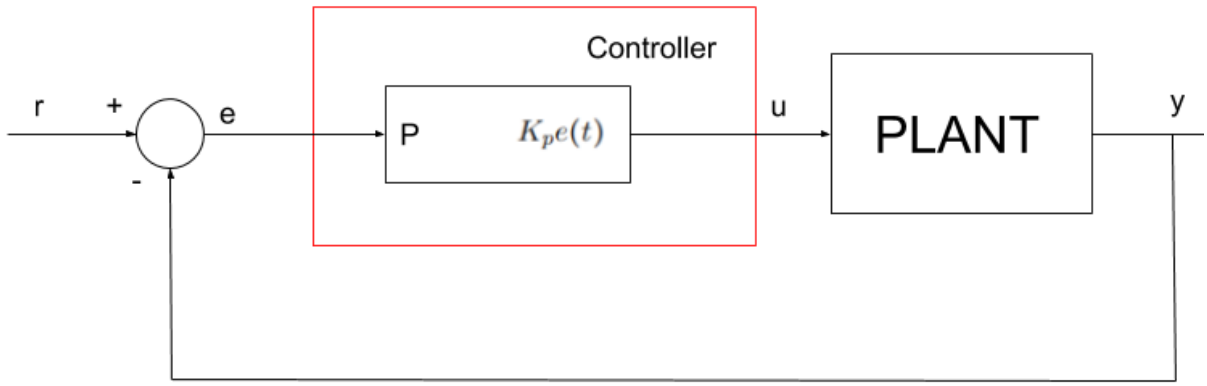



Figure 3.4: P-controller

The output of a P-controller, which is equal to the control input to the plant, is calculated in the time domain and Laplace domain from the feedback error as follows

$$P = K_p e(t) \tag{3.44}$$

A P-controller is mostly used to stabilize an unstable system. From Table 3.2, we can see that if we increase the proportional gain, the steady state error of the system will increase, but a P-controller will never eliminate the steady state error. We can use this controller only when our system is tolerable to a constant steady state error. Increasing the proportional gain also decrease the rise time and leads to overshoot.

### 3.3.3 PI-controller

A PI-controller is a controller consisting of a proportional gain and an integral gain, as shown in Figure 3.5.

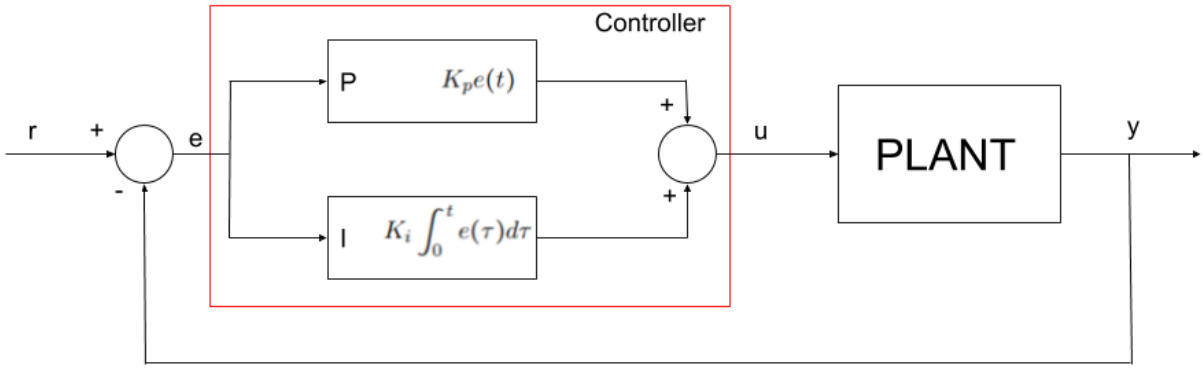



Figure 3.5: PI-controller

The output of a PI-controller, which is equal to the control input to the plant, is calculated in the time domain from the feedback error as follows

$$PI = K_p e(t) + K_i \int_0^t e(\tau)d\tau = K_p(e(t) + \frac{1}{T_i s} \int_0^t e(\tau)d\tau) \tag{3.45}$$

In Laplace domain the output, and transfer function, of the controller is given by

$$PI(s) = K_p + \frac{K_i}{s} = \frac{K_p s + K_i}{s} = K_p(1 + \frac{1}{T_i s}) \tag{3.46}$$

A PI-controller is especially used to eliminate the steady state error from the P-controller. But the integral term has a negative impact of the speed of the response and stability of the system, and is therefore often used when speed of the system response is not a problem.

### 3.3.4 PD-controller

A PD-controller is a controller consisting of a proportional gain and a derivative gain, as shown in Figure 3.6.

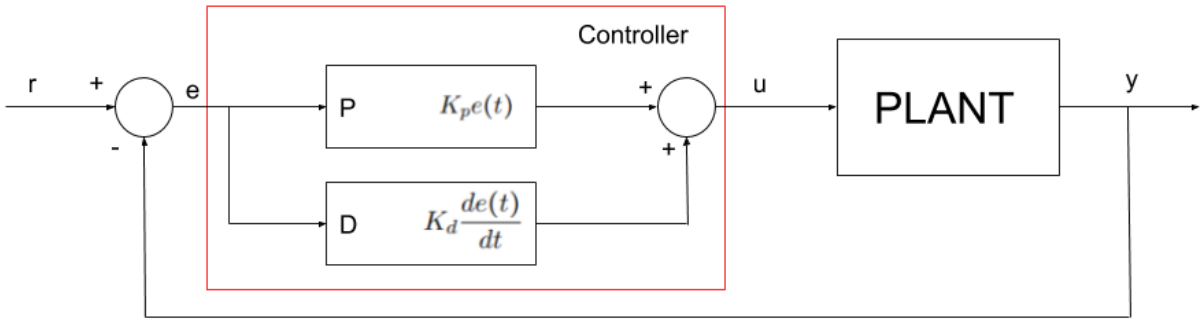



Figure 3.6: PD-controller

The output of a PD-controller, which is equal to the control input to the plant, is calculated in the time domain from the feedback error as follows

$$PD = K_p e(t) + K_d \frac{de(t)}{dt} = K_p(e(t) + T_d \frac{de(t)}{dt}) \tag{3.47}$$

In Laplace domain the output, and transfer function, of the controller is given by

$$PD(s) = K_p + K_d s = \frac{K_d s^2 + K_p s}{s} = K_p(1 + T_d s) \tag{3.48}$$

Since the derivative term of the controller has the ability to predict future errors of the system response, the intention of using a PD-controller is to increase the stability of the system. The derivative is taken from the output response of the system instead of the error signal, in order to avoid the effect of sudden change of the error signal, but the derivative part of this controller cannot be used alone because it amplifies the system noise.

### 3.3.5 Ziegler-Nichols Tuning

To obtain a desired behavior for a system, it is necessary to adjust the controller parameters. This is called tuning the system. There are several ways to tune a system. For instance, a simple method is to connect a controller, increase the gain until the system starts to oscillate, and then reduce the gains by an appropriate factor. Another method is to determine the controller parameters based on open-loop response measurements.

One of the most commonly used tuning rules is the Ziegler-Nichols method. In the 1940s Ziegler and Nichols developed two techniques for controller tuning: Ziegler-Nichols open-loop tuning and Ziegler-Nichols closed-loop tuning. The idea for both tuning methods was to make a simple experiment, extract some features for the experimental data of the system dynamics, and then determine the controller parameters from these features.

The open-loop method is based on the open-loop step response of the system, where the step response is measured by applying a step input to the system and recording the response. Using Ziegler-Nichols open-loop tuning method with dead time $L$, reaction rate $R$ and amplitude $U$ of step input, the controller parameters for a P-, PI- and PID-controller is given in Table 3.3.

Table 3.3: Ziegler-Nichols open-loop controller parameters

| **Type** | $K_p$ | $T_i = \frac{K_p}{K_i}$ | $T_d = \frac{K_d}{K_p}$ |
|---|---|---|---|
| **P** | $\frac{1}{LR/U}$ | $\infty$ | 0 |
| **PI** | $\frac{0.9}{LR/U}$ | $3.3L$ | 0 |
| **PID** | $\frac{1.2}{LR/U}$ | 2L | 0.5L |

The closed-loop method is based on direct adjustment of the controller parameters. A controller is connected to the system using only the proportional gain. This gain will be increased until the system starts to oscillate, and the value of the proportional gain, when the system starts to oscillate, is called the ultimate gain $K_u$, while the period of the oscillation is called the ultimate time $P_u$. Using Ziegler-Nichols closed-loop tuning method with ultimate gain $K_u$ and ultimate period $P_u$, the controller parameters for a P-, PI-,PD- and PID-controller are given in Table 3.4.

Table 3.4: Ziegler-Nichols closed-loop controller parameters

| **Type** | $K_p$ | $T_i = \frac{K_p}{K_i}$ | $T_d = \frac{K_d}{K_p}$ |
|---|---|---|---|
| **P** | $0.5K_u$ | $\infty$ | 0 |
| **PI** | $0.45K_u$ | $\frac{P_u}{1.2}$ | 0 |
| **PD** | $0.8K_u$ | $\infty$ | $\frac{P_u}{8}$ |
| **PID** | $0.6K_u$ | $\frac{P_u}{2}$ | $\frac{P_u}{8}$ |

### 3.3.6 LQR

LQRs have been widely used in many control system designs due to its stability and robustness. The LQR design consists of a state feedback controller, which will minimize the objective function J, given in Equation (3.50). A feedback gain matrix is designed to obtain some agreements

between the use of control effort, the magnitude and the speed of the response, which will ensure that the system will be stable. A LQR is built relaying on state-space methods, which is a method about using state variables to describe a dynamic system by a set of first-order differential equations, instead of using nth-order differential equations.

To design this controller, the system must be controllable. According to Nise's CONTROL SYSTEM ENGINEERING [16], the definition of a controllable system is: *"If an input to a system can be found that takes every state variable from a desired initial state to the desired finale state, the system is said to be controllable; otherwise, the system is uncontrollable"*. This means that to control the system each state variable can be changed by changing the input signal. The input signal must be able to control all the state variables, and if any of the state variables can not be controlled by the input signal, then the system is uncontrollable.

Figure 3.7 shows a block diagram of a LQR, which can be designed on the state-space form given in Equation (3.49).

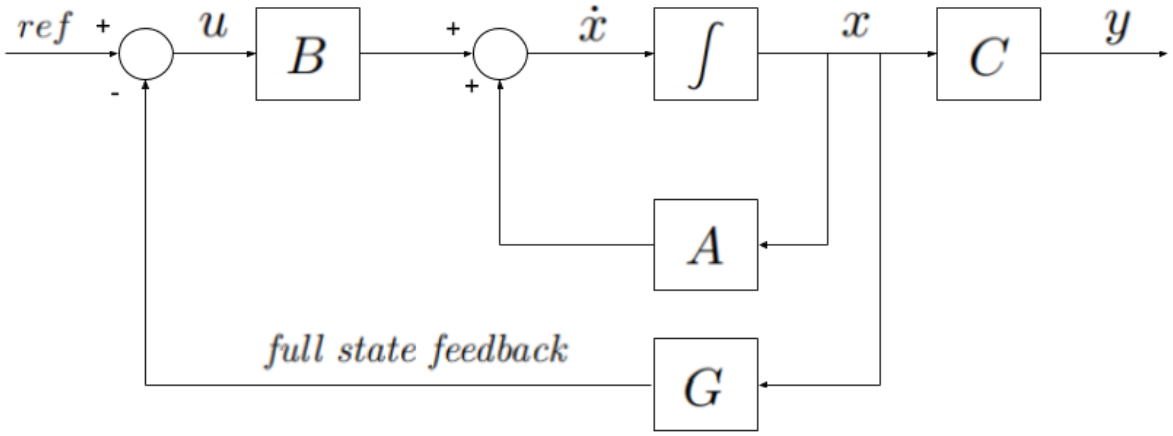



Figure 3.7: Linear-Quadratic Optimal Set-point Regulation

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \tag{3.49}$$

The performance index for such a controller is

$$J = \min_u \left\{ \frac{1}{2} \int_0^T (y^T Q y + u^T R u) dt = \frac{1}{2} \int_0^T (x^T C^T Q C x + u^T R u) dt \right\} \tag{3.50}$$

where the Design weights are

$$\begin{aligned} Q = Q^T \geq 0 \quad \text{(output weight)} \\ R = R^T > 0 \quad \text{(input weight)} \end{aligned}$$

and the optimal solution is

$$\begin{aligned} u &= -R^{-1} B^T P_\infty x = Gx \\ P_\infty + A^T P_\infty - P_\infty B R^{-1} B^T P_\infty + C^T Q C &= 0 \end{aligned} \tag{3.51}$$

where the Algebraic Riccati equation is

$$P = P^T > 0 \tag{3.52}$$

If the controller must track a time-varying reference trajectory, the LQR can be redesign as shown in Figure 3.8.

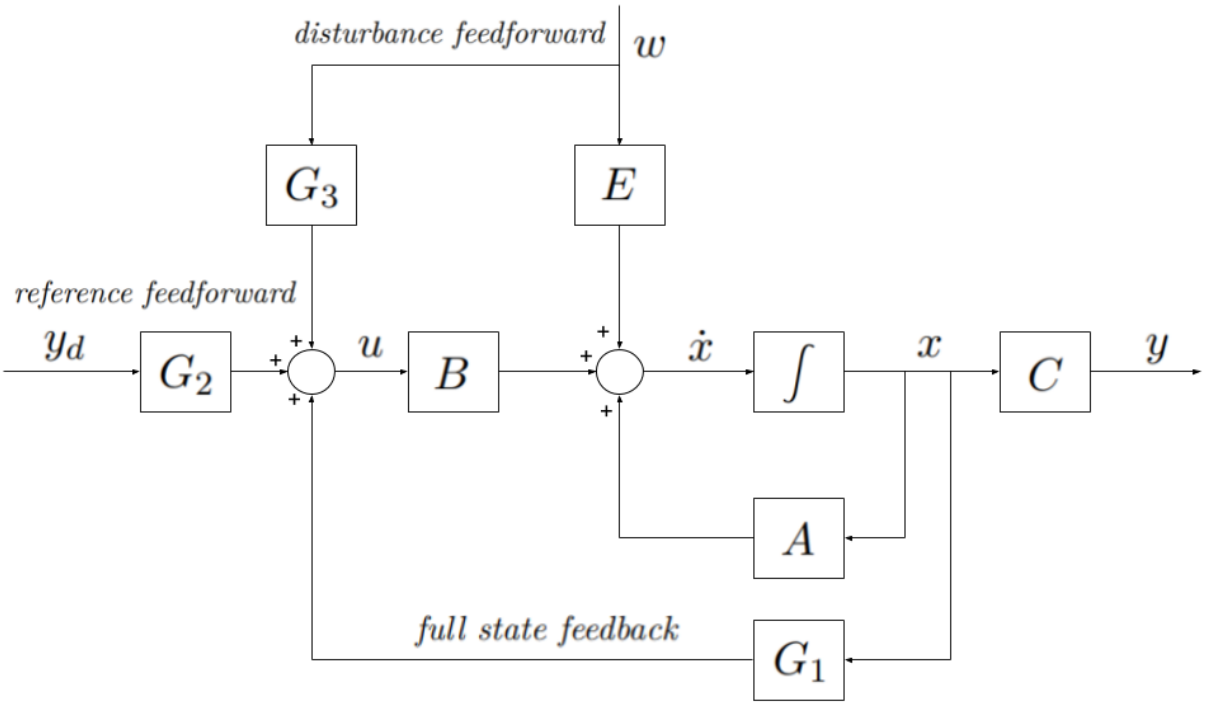



Figure 3.8: Linear-Quadratic Optimal Trajectory Tracking Control

Now the controller can be designed on the state-space form given by the equations

$$\begin{aligned} \dot{x} &= Ax + Bu + Ew \\ y &= Cx \end{aligned} \tag{3.53}$$

where

$$w = \quad \text{disturbance to the system}$$

If the case is

$$x_d = \text{constant}, \quad w = \text{constant}, \quad \forall \quad t\epsilon[0, T_1]$$

the general solution for the linear time-invariant system can be written as

$$u = G_1 x + G_2 y_d + G_3 w \tag{3.54}$$

where

$$\begin{aligned} G_1 &= -R^{-1}B^T P_\infty \\ G_2 &= -R^{-1}B^T(A + BG_1)^{-T}C^T Q \\ G_3 &= R^{-1}B^T(A + BG_1)^{-T}P_\infty E \end{aligned} \tag{3.55}$$

### 3.3.7 Inverse Kinematics Methods

It is possible to use algebraic methods to solve the inverse kinematics of a robot manipulator, but the algebraic solution exists only for a restricted class of cases. The joint angles can be expressed, with for instance the DH convention, using the end-effector position. This works for a 2DOF robot manipulator, but since the DOFs for most cases is higher, iterative methods are necessary. These methods solve the kinematic equations using a sequence of steps, which lead to a better solution for the joint angles. The methods are used to minimize the difference between the desired and current position of the end-effector. There are several methods using this technique, and two of them are:

- Jacobian Inversion Method
- Jacobian Transpose Method

For the Jacobian inversion method, the relation between the joint angles and the end-effector position can be expressed as

$$\dot{\theta} = J^{-1}(\theta)\dot{X} \tag{3.56}$$

where $\theta$ are the joint angles, $\dot{\theta}$ are the joint velocities, $J^{-1}$ is the inverse Jacobian matrix and $\dot{X}$ are the position of the end-effector in x-, y- and z-direction. Figure 3.9 shows the Jacobian inversion method in form of a block diagram.

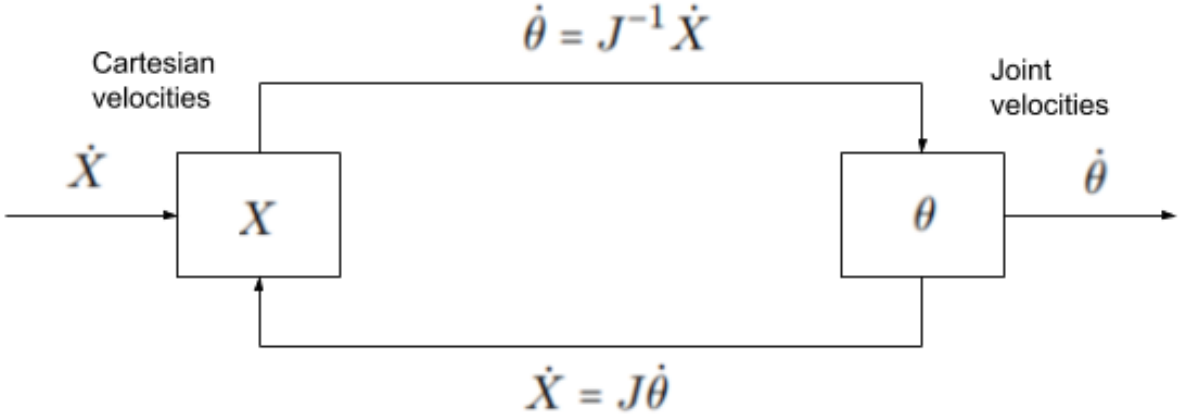



Figure 3.9: Jacobian inversion method

One concern with this method is that using the inverse Jacobian matrix may not lead to one solution, but an infinite number of solutions, and singularities usually occur. Using the transpose of the Jacobian, instead of the inverse, removes the singularity problems significantly.

For the Jacobian transpose method, the relation between the force $F$ and the generalized forces $\tau$ is expressed as

$$\tau = J^T F \tag{3.57}$$

where $J^T$ is the transpose Jacobian matrix.

The generalized forces can be expresses either with the joint variable accelerations $\ddot{\theta}$ or joint velocities $\dot{\theta}$. Because this method is not interested in the dynamic behavior of the system, only the joint velocities are used for the necessity of this method, and the relation between the force and the joint velocities can be expressed as

$$\dot{\theta} = J^T F \tag{3.58}$$

Figure 3.10 shows the Jacobian transpose method in form of a block diagram.

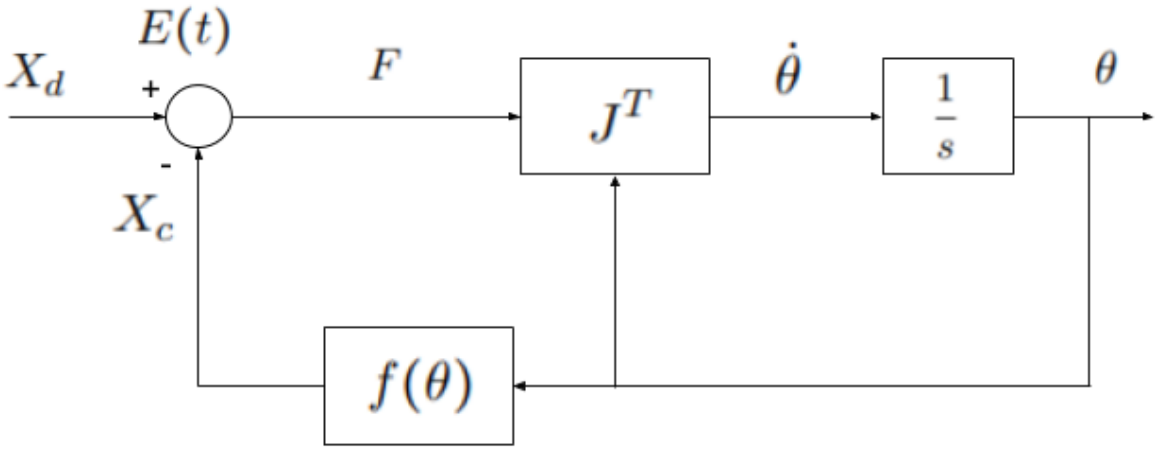



Figure 3.10: Jacobian transpose method

The force $F$ corresponds to the error $E(t)$, which is expressed as

$$E(t) = X_d - X_c \tag{3.59}$$

and is the difference between the desired end-effector trajectory and the current end-effector position. $f(\theta)$ describes the forward kinematics from the joint angles to the end-effector position.

# Chapter 4

# Description of the Crane

Figure 4.1 shows an AUTOCAD-drawing of an assembly of the crane, mounted to the deck of the vessel. This drawing is provided from MacGregor in conjunction with the last year master thesis.

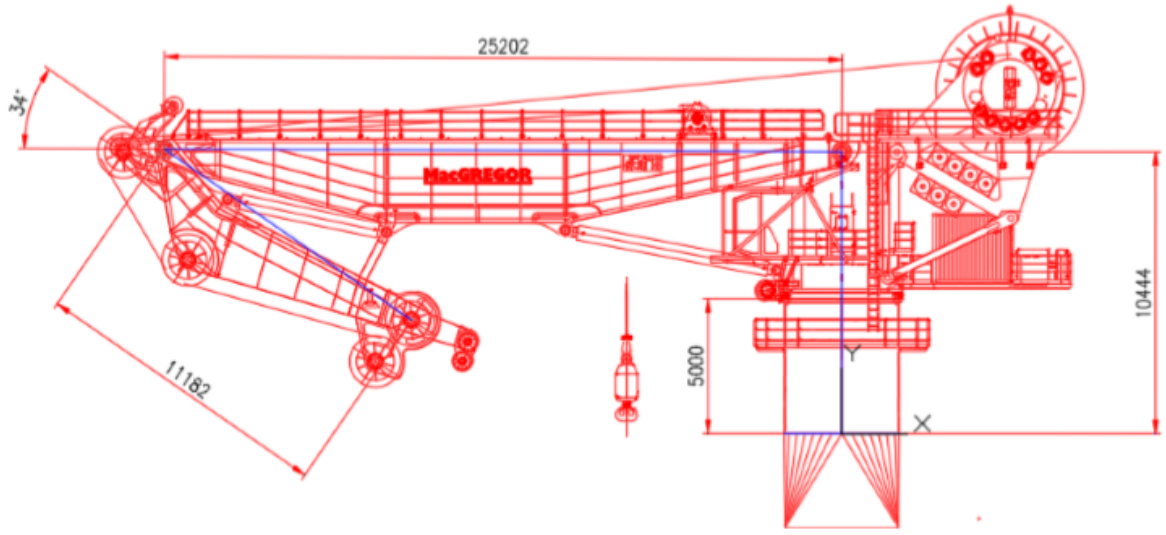



Figure 4.1: Assembly of the crane

From the figure we can see that the assembly of the crane consists of a great number of bodies, and for that reason some simplifications and approximations are necessary to obtain a viable system for its purpose, when it comes to modeling, simulation and control of the crane. The idea is to include just the elements that are considerable for its investigation. For our purpose we only need to include bodies which have the most contributing inertia.

Table 4.1 shows the properties of mass and the center of mass for each part of the crane assembly, and are provided by MacGregor.

Table 4.1: Measurements of the crane assembly

| **Part** | $m[tonne]$ | $\overline{x}[m]$ | $\overline{z}[m]$ |
|---|---|---|---|
| Foundation | 20.00 | 0.00 | 2.50 |
| King | 88.90 | -0.5 | 7.24 |
| Winch | 173.10 | -0.95 | 12.27 |
| Wire | 149.94 | -6.50 | 13.00 |
| Main cylinders | 26.70 | 6.36 | 6.79 |
| Knucke Jib Cylinders | 12.40 | 19.87 | 8.11 |
| Main Jib | 47.90 | 11.73 | 9.85 |
| Knucke Jib | 36.00 | 21.22 | 6.53 |
| Misc 1 | 3.10 | 1.70 | 9.75 |
| Misc 2 | 2.30 | -4.07 | 6.91 |
| **Sum** | 560.34 | - | - |

To assemble a simplified model of the crane, some of the bodies can be combined. It is possible to use four bodies describing the whole crane. Figure 4.2 shows the simplified model with the four combined bodies, and how the individual parts are combined is listed below.

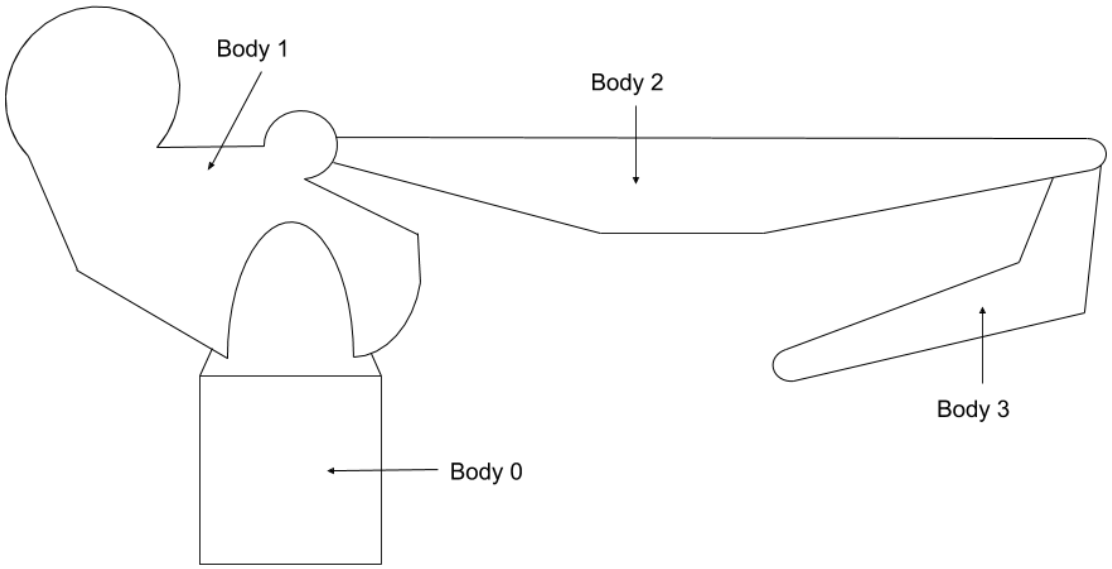



Figure 4.2: Simplified crane model with combined bodies

- Body 0: Foundation
- Body 1: King assembly, Main Winch, Wire, Main Cylinders, Misc 1 and Misc 2
- Body 2: Main Jib and Knuckle-jib Cylinders
- Body 4: Knuckle Jib

Then the mass and center of mass of each of the four bodies are defined by adding the individual component each body consist of together, as presented in Table 4.2.

Table 4.2: Measurements of the crane assembly with combined bodies

| **Part** | $m$ [tonne] | $\overline{x}$ [m] | $\overline{z}$ [m] |
|---|---|---|---|
| Body 0/Foundation | 20.00 | 0.00 | 2.50 |
| Body 1 | 444.04 | -4.24 | 11.14 |
| Body 2 | 60.34 | 13.40 | 9.49 |
| Body 3 | 36.00 | 21.22 | 6.53 |
| **Sum** | 560.34 | - | - |

# Chapter 5

# Modelling of Crane

This chapter is about modelling of the crane, both the mathematical model describing the crane and the crane design in Simulink. First, a mathematically model relying on crane kinematics and then crane dynamics will be developed. Finally, the crane is designed in Simulink based on the crane dynamics using the equations of motion.

## 5.1 Crane Kinematics

Kinematic equations are used to describe the relation between the individual joints of the crane and the position and orientation of the end-effector. This includes the forward kinematics, where the DH convention allow us to model the crane as an open chain. It also includes inverse kinematics, which is used to find the joint angles from the end-effector. The crane needs to be modelled using the DH convention, where the geometric Jacobian matrices between Frame 3 and the joints with respect to the velocity and acceleration are found as well. To accomplish the kinematic model, the relation between the cylinder stroke and the joint angles is found for both cylinders. Equations related to the crane kinematics are calculated from the equations in chapter 3.1 and are further based on [11], where some modifications are done.

### 5.1.1 DH Convention

The crane joints are modelled as an open chain using the DH convention. Figure 5.1 represent the crane consisting of three links connected by three joints. The foundation of the crane is fixed, and joint 1 is placed in the center of the intersection between body 0 and body 1. Therefore, the local base-frame is located in joint 1, also shown in the Figure 5.1. Joint 1 is connecting link 1 to the base-frame, joint 2 is connecting link 1 to link 2, and joint 3 is connecting link 2 to link 3.

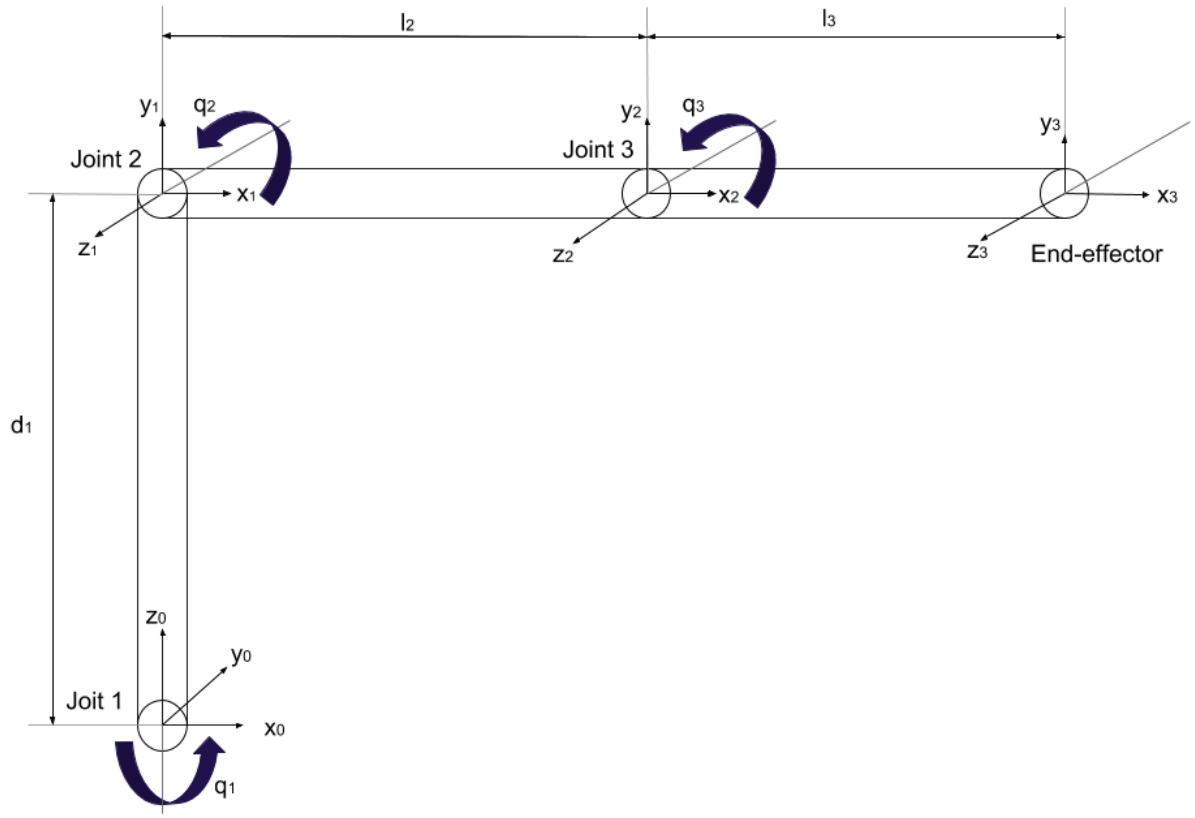



Figure 5.1: DH representation of the crane

Table 5.1 shows the DH parameters for the crane in Figure 5.1.

Table 5.1: DH parameters

| **Link** | $\boldsymbol{a_i}$ | $\boldsymbol{\alpha_i}$ | $\boldsymbol{d_i}$ | $\boldsymbol{\theta_i}$ |
|---|---|---|---|---|
| 1 | 0 | $\frac{\pi}{2}$ | $d_1$ | $\theta_1^*$ |
| 2 | $l_2$ | 0 | 0 | $\theta_2^*$ |
| 3 | $l_3$ | 0 | 0 | $\theta_3^*$ |

From the table, the variable angles $\theta_1^*$, $\theta_2^*$ and $\theta_3^*$ correspond to the joint angles $q_1$, $q_2$ and $q_3$ relative to the local coordinate system for each joint, where $\theta_1^* = q_1$, $\theta_2^* = q_2$ and $\theta_3^* = q_3$. As seen in Figure 5.1, link 1 has the the angle $\alpha_1 = \frac{\pi}{2}$, which result in a link length of $a_1 = 0$. The length of link 2 and 3 are signed with the variables $l_2$ and $l_3$, respectively. Joint 1 is a prismatic joint and has a linear motion in z-direction with the variable $d_1$. Since joint 2 and 3 are revolute joints the variables $d_2$ and $d_3$ become zero.

The DH procedure determines the transformation of the model from Frame 0 to Frame 3 with four elementary homogeneous transformations, where the resulting transformation matrices are given by

$$\boldsymbol{T}_1^0(q_1) = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{T}_2^1(q_2) = \begin{bmatrix} c_2 & -s_2 & 0 & l_2 c_2 \\ s_2 & c_2 & 0 & l_2 s_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}, \boldsymbol{T}_3^2(q_3) = \begin{bmatrix} c_3 & s_3 & 0 & a_3 c_3 \\ s_3 & c_3 & 0 & a_3 s_3 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.1}$$

where

$$c_1 = cos(q_1), \quad s_1 = sin(q_1)$$

$$c_2 = cos(q_2), \quad s_2 = sin(q_2)$$

$$c_3 = cos(q_3), \quad s_3 = sin(q_3)$$

Then the total transformation from Frame 0 to Frame 3 becomes

$$\boldsymbol{T}_3^0(q) = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & s_1 & c_1(l_2c_2 + l_3c_{23}) \\ s_1c_{23} & -s_1s_{23} & -c_1 & s_1(l_2c_2 + l_3c_{23}) \\ s_{23} & c_{23} & 0 & d_1 + l_2s_2 + l_3s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{5.2}$$

where

$$c_{23} = cos(q_2 + q_3), \quad s_{23} = sin(q_2 + q_3)$$

### 5.1.2 Geometric Jacobian between Frame 3 and Joints

The geometric Jacobian between the end-effector (Frame 3) needs to be found to determine the joint velocities and accelerations that correspond to a trajectory of the end-effector. This trajectory can be found in chapter 6.

#### Velocity

The first step to find the geometric Jacobian is to find the transformation from Frame 0 to Frame 3. To obtain the geometric Jacobian from Frame 0 to 1, Frame 0 to 2 and Frame 0 to 3, the homogeneous transformation matrices $\boldsymbol{H}_i^0$ are needed.

Frame 0 to Frame 1:

$$\boldsymbol{H}_1^0 = \boldsymbol{T}_1^0(q_1) \tag{5.3}$$

$$\boldsymbol{H}_1^0 = \begin{bmatrix} c_1 & 0 & s_1 & 0 \\ s_1 & 0 & -c_1 & 0 \\ 0 & 1 & 0 & d_1 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From Frame 0 to Frame 2:

$$\boldsymbol{H}_2^0 = \boldsymbol{T}_1^0(q_1)\boldsymbol{T}_2^1(q_2) \tag{5.4}$$

$$\boldsymbol{H}_2^0 = \begin{bmatrix} c_1c_2 & -c_1s_2 & s_1 & l_2c_1c_2 \\ s_1c_2 & -s_1c_2 & -c_1 & l_2s_1c_2 \\ s_2 & c_2 & 0 & d_1 + l_2s_2 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

From Frame 0 to Frame 3:

$$\boldsymbol{H}_3^0 = \boldsymbol{T}_1^0(q_1)\boldsymbol{T}_2^1(q_2)\boldsymbol{T}_3^2(q_3) \tag{5.5}$$

$$\boldsymbol{H}_3^0 = \begin{bmatrix} c_1c_{23} & -c_1s_{23} & s_1 & c_1(l_2c_2 + l_3c_{23}) \\ s_1c_{23} & -s_1s_{23} & -c_1 & s_1(l_2c_2 + l_3c_{23}) \\ s_{23} & c_{23} & 0 & d_1 + l_2s_2 + l_3s_{23} \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

The next step is to find the vectors $\mathbf{r}_i$ and $\mathbf{z}_{i-1}$, which easily can be found from the transformation matrices. This results in the following $\mathbf{r}_i$ vectors

$$\mathbf{r}_0 = \begin{bmatrix} 0 \\ 0 \\ 0 \end{bmatrix}, \quad \mathbf{r}_1 = \begin{bmatrix} 0 \\ 0 \\ d_1 \end{bmatrix}, \quad \mathbf{r}_2 = \begin{bmatrix} l_2c_1c_2 \\ l_2s_1c_2 \\ d_1 + l_2s_2 \end{bmatrix}, \quad \mathbf{r}_3 = \begin{bmatrix} c_1(l_2c_2 + l_3c_{23} \\ s_1(l_2c_2 + l_3c_{23} \\ d_1 + l_2s_2 + l_3s_{23} \end{bmatrix} \tag{5.6}$$

and the following $\mathbf{z}_{i-1}$ vectors

$$\mathbf{z}_0 = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}, \quad \mathbf{z}_1 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix}, \quad \mathbf{r}_2 = \begin{bmatrix} s_1 \\ -c_1 \\ 0 \end{bmatrix} \tag{5.7}$$

Since the geometric Jacobian is given by

$$\boldsymbol{J}_i = \begin{bmatrix} \mathbf{z}_{i-1} \times (\mathbf{r}_n - \mathbf{r}_{i-1}) \\ \mathbf{z}_{i-1} \end{bmatrix} \tag{5.8}$$

the vector $\mathbf{r}_3 - \mathbf{r}_{i-1}$ needs to be calculated, and is given by

$$\mathbf{r}_3 - \mathbf{r}_0 = \begin{bmatrix} c_1(l_2c_2 + l_3c_{23}) \\ s_1(l_2c_2 + l_3c_{23}) \\ d_1 + l_2s_2 + l_3s_{23} \end{bmatrix} \tag{5.9}$$

$$\mathbf{r}_3 - \mathbf{r}_1 = \begin{bmatrix} c_1(l_2c_2 + l_3c_{23}) \\ s_1(l_2c_2 + l_3c_{23}) \\ l_2s_2 + l_3s_{23}^* \end{bmatrix} \tag{5.10}$$

$$\mathbf{r}_3 - \mathbf{r}_2 = \begin{bmatrix} l_3c_1c_{23} \\ l_3s_1c_{23} \\ l_3s_{23} \end{bmatrix} \tag{5.11}$$

Then the cross product $\mathbf{z}_{i-1} \times (\mathbf{r}_e - \mathbf{r}_{i-1})$ can be calculated for all joint variables.

For joint variable 1:

$$z_0 \times (\mathbf{r}_3 - \mathbf{r}_0) = \begin{bmatrix} -s_1(l_2c_2 + l_3c_{23}) \\ c_1(l_2c_2 + l_3c_{23}) \\ 0 \end{bmatrix} \tag{5.12}$$

For joint variable 2:

$$z_1 \times (\mathbf{r}_3 - \mathbf{r}_1) = \begin{bmatrix} -c_1(l_2c_2 + l_3s_{23}) \\ -s_1(l_2s_2 + l_3s_{23}) \\ l_2c_2 + l_3c_{23} \end{bmatrix} \tag{5.13}$$

For joint variable 3:

$$z_2 \times (\mathbf{r}_3 - \mathbf{r}_2) = \begin{bmatrix} -l_3c_1c_{23} \\ -l_3s_1s_{23} \\ l_3c_{23} \end{bmatrix} \tag{5.14}$$

Then the geometric Jacobian for the joints are given as follows

Joint variable 1:

$$\boldsymbol{J}_1 = \begin{bmatrix} -s_1(l_2c_2 + l_3c_{23}) \\ c_1(l_2c_2 + l_3c_{23}) \\ 0 \\ 0 \\ 0 \\ 1 \end{bmatrix} \tag{5.15}$$

Joint variable 2:

$$\boldsymbol{J}_2 = \begin{bmatrix} -c_1(l_2s_2 + l_3s_{23}) \\ -s_1(l_2s_2 + l_3s_{23}) \\ l_2c_2 + l_3c_{23} \\ s_1 \\ -c_1 \\ 0 \end{bmatrix} \tag{5.16}$$

Joint variable 3:

$$\boldsymbol{J}_3 = \begin{bmatrix} -l_3c_1c_{23} \\ -l_3s_1s_{23} \\ l_3c_{23} \\ s_1 \\ -c_1 \\ 0 \end{bmatrix} \tag{5.17}$$

Finally, the total geometric Jacobian can be written as

$$\boldsymbol{J} = \begin{bmatrix} \boldsymbol{J}_v \\ \boldsymbol{J}_\omega \end{bmatrix} = \begin{bmatrix} \boldsymbol{J}_1 & \boldsymbol{J}_2 & \boldsymbol{J}_3 \end{bmatrix} = \begin{bmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2s_2 + l_3s_{23}) & -l_3c_1c_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2s_2 + l_3s_{23}) & -l_3s_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \\ 0 & s_1 & s_1 \\ 0 & -c_1 & -c_1 \\ 1 & 0 & 0 \end{bmatrix} \tag{5.18}$$

The Jacobian matrix that is used to determine the joint velocities corresponding to the trajectory that is explained in chapter 6, is the first three rows of matrix in Equation (5.18), and is given by

$$\boldsymbol{J} = \begin{bmatrix} -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2s_2 + l_3s_{23}) & -l_3c_1c_{23} \\ c_1(l_2c_2 + l_3c_{23}) & -s_1(l_2s_2 + l_3s_{23}) & -l_3s_1s_{23} \\ 0 & l_2c_2 + l_3c_{23} & l_3c_{23} \end{bmatrix} \tag{5.19}$$

**Acceleration**

Using Equation (3.19) the derivative of the Jacobian for each joints is

$$\frac{d}{dt}\boldsymbol{J}(\mathbf{q}_1) = \frac{\partial \boldsymbol{J}}{\partial q_1} = \begin{bmatrix} -c_1(l_2c_2 + l_3c_{23} & s_1(l_2s_2 + l_3s_{23} & l_3s_1s_{23} \\ -s_1(l_2c_2 + l_3c_{23}) & -c_1(l_2s_2 + l_3s_{23}) & -l_3c_1s_{23} \\ 0 & 0 & 0 \\ 0 & c_1 & c_1 \\ 0 & s_1 & s_1 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.20}$$

$$\frac{d}{dt}\boldsymbol{J}(\boldsymbol{q}_2) = \frac{\partial \boldsymbol{J}}{\partial q_2} = \begin{bmatrix} -s_1(-l_2s_2 - l_3s_{23} & -c_1(l_2c_2 + l_3c_{23} & -l_3c_1c_{23} \\ c_1(-l_2s_2 - l_3s_{23}) & -s_1(l_2c_2 + l_3c_{23}) & -l_3s_1c_{23} \\ 0 & -l_2s_2 - l_3s_{23} & -l_3s_{23} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.21}$$

$$\frac{d}{dt}\boldsymbol{J}(\boldsymbol{q}_3) = \frac{\partial \boldsymbol{J}}{\partial q_3} = \begin{bmatrix} l_3s_1s_{23} & -l_3c_1c_{23} & -l_3c_1c_{23} \\ -l_3c_1s_{23} & l_3c_1c_{23} & -l_3s_1c_{23} \\ 0 & -l_3s_{23} & -l_3s_{23} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} \tag{5.22}$$

Finally, the total derivative of the Jacobian is then given by

$$\frac{d}{dt}\boldsymbol{J}(\mathbf{q}) = \frac{\partial \boldsymbol{J}}{\partial q_1}\dot{q}_1 + \frac{\partial \boldsymbol{J}}{\partial q_2}\dot{q}_2 + \frac{\partial \boldsymbol{J}}{\partial q_3}\dot{q}_3 \tag{5.23}$$

and the first three rows of the matrix are used to determine the joint accelerations corresponding to the same trajectory.

### 5.1.3 Inverse Kinematics

Inverse kinematics is used to find the joint angles that correspond to the trajectory of the end-effector, described in chapter 6. These inverse kinematic equations are based on [10] and are modified. The equations are also simplified since link 1 is only moving along the z-axis.

The first joint angle can be calculated as

$$q_1 = tan^{-1}\left(\frac{y_e}{x_e}\right) \tag{5.24}$$

The trigonometric relations for joint angle 3 can be expresses as

$$c_3 = \frac{x_e^2 + y_e^2 + (z_e - d_1)^2 - l_2^2 - l_3^2}{2l_2l_3} \tag{5.25}$$

and

$$s_3 = \sqrt{1 - c_3^2}) \tag{5.26}$$

Then the third joint angle is given by

$$q_3 = tan^{-1}\left(\frac{s_3}{c_3}\right) \tag{5.27}$$

.

Finally, the second joint angle can be expressed with the use of $s_3$ and $c_3$, and is given by

$$q_2 = tan^{-1}\left(\frac{z_e - d_1}{\sqrt{x_e^2 + y_e^2}}\right) - tan^{-1}\left(\frac{l_3 s_3}{l_2 + l_3 c_3}\right) \tag{5.28}$$

### 5.1.4 Actuator Kinematics

Actuator kinematics describes the relations between cylinder strokes and joint angles. Equations related to these relations are based on [5], but some modifications are done. Figure 5.2 shows trigonometric relations between cylinder 1 and joint 2, while figure 5.3 shows relations between cylinder 2 and joint 3.

Cylinder 1:

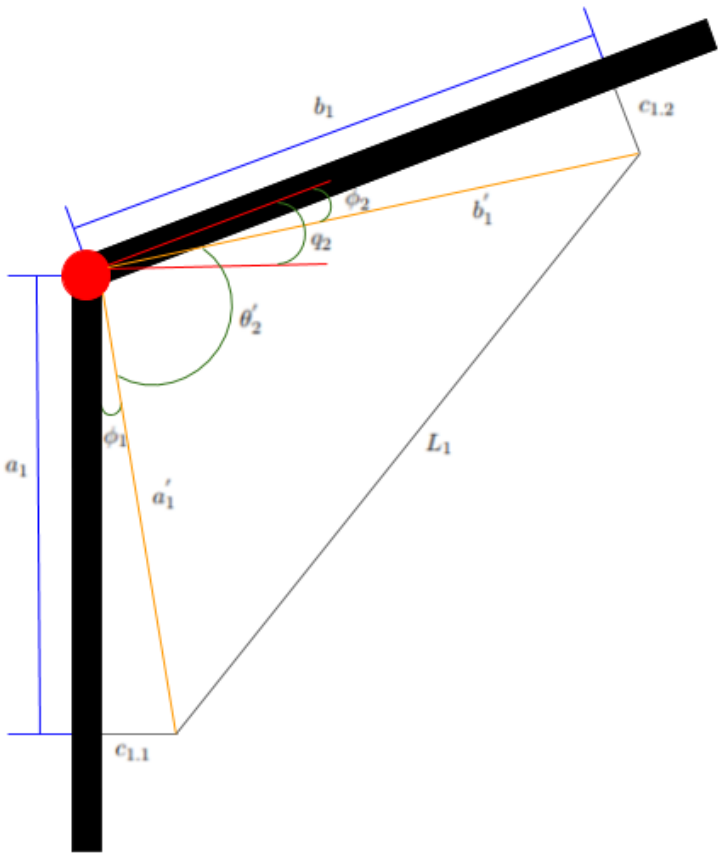



Figure 5.2: Trigonometric relations between cylinder 1 and joint 2

By [5], using the rule of cosine we obtain the following equation for the cylinder 1

$$L_1 = \sqrt{a_1'^2 + b_1'^2 - 2a_1'b_1'cos(\theta_2')} \tag{5.29}$$

where the angle $\theta_2'$ is given by

$$\theta_2' = q_2 - \phi_1 - \phi_2 + \frac{\pi}{2} \tag{5.30}$$

and the angles $\phi_1$ and $\phi_2$ are given by

$$\phi_1 = tan^{-1}\left(\frac{c_{1.1}}{a_1}\right) \tag{5.31}$$

$$\phi_2 = tan^{-1}\left(\frac{c_{1.2}}{b_1}\right) \tag{5.32}$$

where

$$\begin{aligned} a_1 &= -4.390m \\ b_1 &= 10.2765m \\ c_{1.1} &= 2.435m \\ c_{1.2} &= -2.900m \end{aligned} \tag{5.33}$$

$$a_1' = \sqrt{c_{1.1}^2 + a_1'^2} \tag{5.34}$$

$$b_1' = \sqrt{b_1'^2 + c_{1.2}^2} \tag{5.35}$$

Cylinder 2:

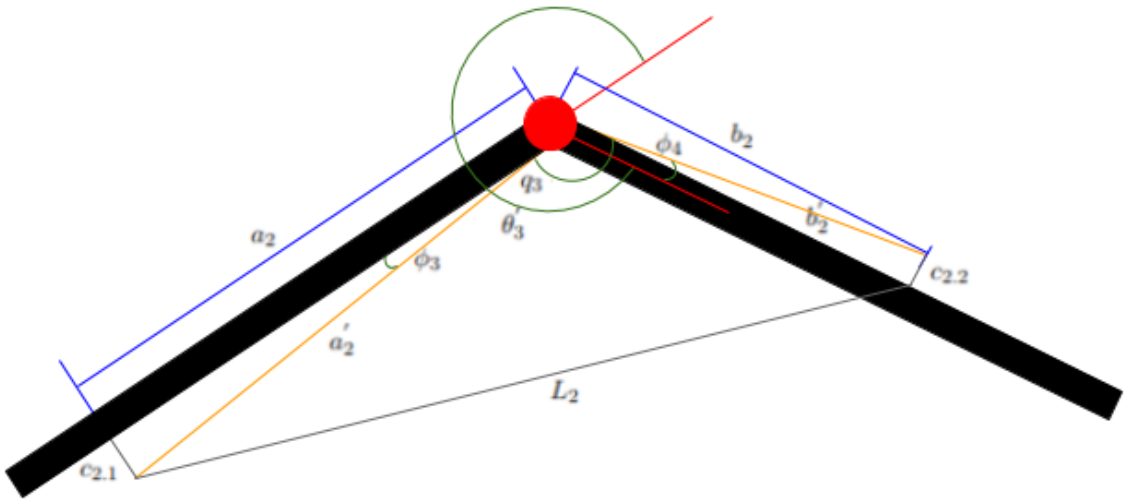



Figure 5.3: Trigonometric relations between cylinder 2 and joint 3

Again by [5], using the rule of cosine we obtain the following equation for cylinder 2

$$L_2 = \sqrt{a_2'^2 + b_2'^2 - 2a_2'b_2'cos(\theta_3')} \tag{5.36}$$

where the angle $\theta_3^{'}$ is given by

$$\theta_3^{'} = q_3 - \phi_3 + \phi_4 - \pi \tag{5.37}$$

and the angles $\phi_3$ and $\phi_4$ are given by

$$\phi_3 = tan^{-1}\left(\frac{c_{2.1}}{a_2}\right) \tag{5.38}$$

$$\phi_4 = tan^{-1}\left(\frac{c_{2.2}}{b_2}\right) \tag{5.39}$$

where

$$\begin{aligned} a_2 &= -8.275m \\ b_2 &= -2.38134m \\ c_{2.1} &= -3.090m \\ c_{2.2} &= -1.8639m \end{aligned} \tag{5.40}$$

$$a_2^{'} = \sqrt{c_{2.1}^2 + a_2^{'2}} \tag{5.41}$$

$$b_2^{'} = \sqrt{b_2^{'2} + c_{2.2}^2} \tag{5.42}$$

The values of the parameters used in these equations, listed in Equation (5.33) and (5.40), are collected from the Master thesis "MODELLING AND SIMULATION OF A KNUCKLE BOOM CRANE AND MARINE CRAFT" [11], which were found from using a measure-tool in AutoCAD.

## 5.2 Crane Dynamics

Dynamic equations are used to describe dynamic behavior of the crane and rely on the kinetic and potential energy of the crane. Equations related to the crane kinematics are calculated from the equations in chapter 3.2 and are further based on [2], where some modification are done.

The crane dynamics can be expressed with the equation of motion, which is given by

$$\boldsymbol{M(q)\ddot{q} + C(\dot{q}, q)\dot{q} + g(q) = \tau} \tag{5.43}$$

The content of the equations of motion are the kinetic energy consisting of the Inertia matrix $\boldsymbol{M(q)}$ and Coriolis and centripetal matrix $\boldsymbol{C(\dot{q}, q)}$, and the potential energy consisting of the gravity vector $\boldsymbol{g(q)}$. $\boldsymbol{\tau}$ is the joint torque vector, which is the input to the dynamic crane model.

### 5.2.1 Kinetic Energy

Using Equation (3.28), the elements of Inertia matrix for the crane are written as

$$\boldsymbol{M(q)} = \begin{bmatrix} M_{11} & M_{12} & M_{13} \\ M_{21} & M_{22} & M_{23} \\ M_{31} & M_{32} & M_{33} \end{bmatrix} \tag{5.44}$$

where

$$
\begin{aligned}
M_{11} &= I1y + I_{2x}s_2^2 + I_{2y}c_2^2 + I_{3x}s_{23}^2 + I3yc_{23}^2 + m_2\Big(\frac{1}{2}l_2\Big)^2c_2^2 + m_3\left(\frac{1}{2}c_{23} + l_2c_2\right)^2 \\
M_{12} &= M_{21} = 0 \\
M_{13} &= M_{31} = 0 \\
M_{22} &= I_{2z} + I_{3z} + m_2\left(\Big(\frac{1}{2}l_2\Big)^2\right) + m_3\left(\Big(\frac{1}{2}l_3\Big)^2 + l_2^2 + l_2l_3c_3\right) \\
M_{23} &= M_{32} = I_{3z} + m_3\left(\Big(\frac{1}{2}l_3\Big)^2 + \frac{1}{2}l_2l_3c_3\right) \\
M_{33} &= I_{3z} + m_3\Big(\frac{1}{2}l_3\Big)^2
\end{aligned}
$$

Further, the Inertia matrix can, by utilizing Equation (3.38) and (3.41), be used to derive the Coriolis and centripetal matrix. Then the elements of this matrix are written as

$$
\boldsymbol{C(q,\dot{q})} = \begin{bmatrix} C_{11} & C_{12} & C_{13} \\ C_{21} & C_{22} & C_{23} \\ C_{31} & C_{32} & C_{33} \end{bmatrix} \tag{5.45}
$$

where

$$
\begin{aligned}
C_{11} &= \frac{1}{2}\left(\frac{\partial M_{11}}{\partial q_2}\dot{q}_2 + \frac{\partial M_{11}}{\partial q_3}\dot{q}_3\right) \\
&= \Big(c_2s_2\big(I_{2x} - I_{2y}\big) + c_{23}s_{23}\big(I_{3x} - I_{3y}\big) - m_2l_2^2c_2s_2 - m_3\big(l_{3c}c_{23} + l_2c_2\big)\big(l_{3c}s_{23} + l_2s_2\big)\Big)\dot{q}_2 \\
&+ \Big(c_{23}s_{23}\big(I_{3x} - I_{3y}\big) - m_3l_{3c}s_{23}\big(l_{3c}c_{23} + l_2c_2\big)\Big)\dot{q}_3 \\
C_{12} &= \frac{1}{2}\frac{\partial M_{11}}{\partial q_2}\dot{q}_1 \\
&= \Big(c_2s_2\big(I_{2x} - I_{2y}\big) + c_{23}s_{23}\big(I_{3x} - I_{3y}\big) - m_2l_{2c}^2c_2s_2 - m_3\big(l_{3c}c_{23} + l_2c_2\big)\big(l_{3c}s_{23} + l_2s_2\big)\Big)\dot{q}_1 \\
C_{13} &= \frac{1}{2}\frac{\partial M_{11}}{\partial q_3}\dot{q}_1 = \Big(c_{23}s_{23}\big(I_{3x} - I_{3y}\big) - m_3l_{3c}s_{23}\big(l_{3c}c_{23} + l_2c_2\big)\Big)\dot{q}_1 \\
C_{21} &= -\frac{1}{2}\frac{\partial M_{11}}{\partial q_2}\dot{q}_2 = -C_{12} \\
C_{22} &= \frac{1}{2}\frac{\partial M_{22}}{\partial q_3}\dot{q}_3 = -\frac{1}{2}m_3l_2l_3s_3\dot{q}_3 \\
C_{23} &= \frac{1}{2}\left(\frac{\partial M_{22}}{\partial q_3}\dot{q}_2 + 2\frac{\partial M_{23}}{\partial q_3}\dot{q}_2\right) = -\frac{1}{2}m_3l_2l_3s_3\dot{q}_2 - m_3l_2l_{3c}s_3\dot{q}_3 \\
C_{31} &= -\frac{1}{2}\frac{\partial M_{11}}{\partial q_3}\dot{q}_1 = -C_{13} \\
C_{32} &= -\frac{1}{2}\frac{\partial M_{22}}{\partial q_3}\dot{q}_2 = \frac{1}{2}m_3l_2l_3s - 3\dot{q}_2 \\
C_{33} &= 0
\end{aligned}
$$

In the Coriolis and centripetal matrix, $l_{ci}$ represents $\frac{l_i}{2}$.

### 5.2.2 Potential Energy

From Equation (3.30), the equation for potential energy for the crane can be written as

$$\boldsymbol{P} = m_1\frac{d_1}{2} + m_2 g\Big(\frac{l_2}{2} + d_1\Big) + m_3 g\Big(\frac{l_3}{2}s_{23} + l_2 s_2 + d_1\Big) \tag{5.46}$$

By utilizing Equation (3.39), the resulting gravity vector for this crane can be written as

$$\boldsymbol{g(q)} = \begin{bmatrix} 0 \\ m_2 g l_{2c} c_2 + m_3 g l_{3c} c_{23} + l_2 c_2 \\ m_3 g l_{3c} c_{23} \end{bmatrix} \tag{5.47}$$

Also in the gravity vector, $l_{ci}$ represents $\frac{l_i}{2}$. The gravity term $g$ is the gravity acceleration in negative direction.

## 5.3 Dynamic Crane Model in Simulink

The equation of motion, Equation (5.43), can be rearranged in the following form

$$\ddot{\boldsymbol{q}} = \boldsymbol{M^{-1}(q)(\tau - C(\dot{q}, q)\dot{q} - g(q))} \tag{5.48}$$

Equation (5.48) can be used to describe the dynamic crane model in form of a block diagram, as shown in Figure 5.4. This dynamic model is based on [2], where some modifications are done.

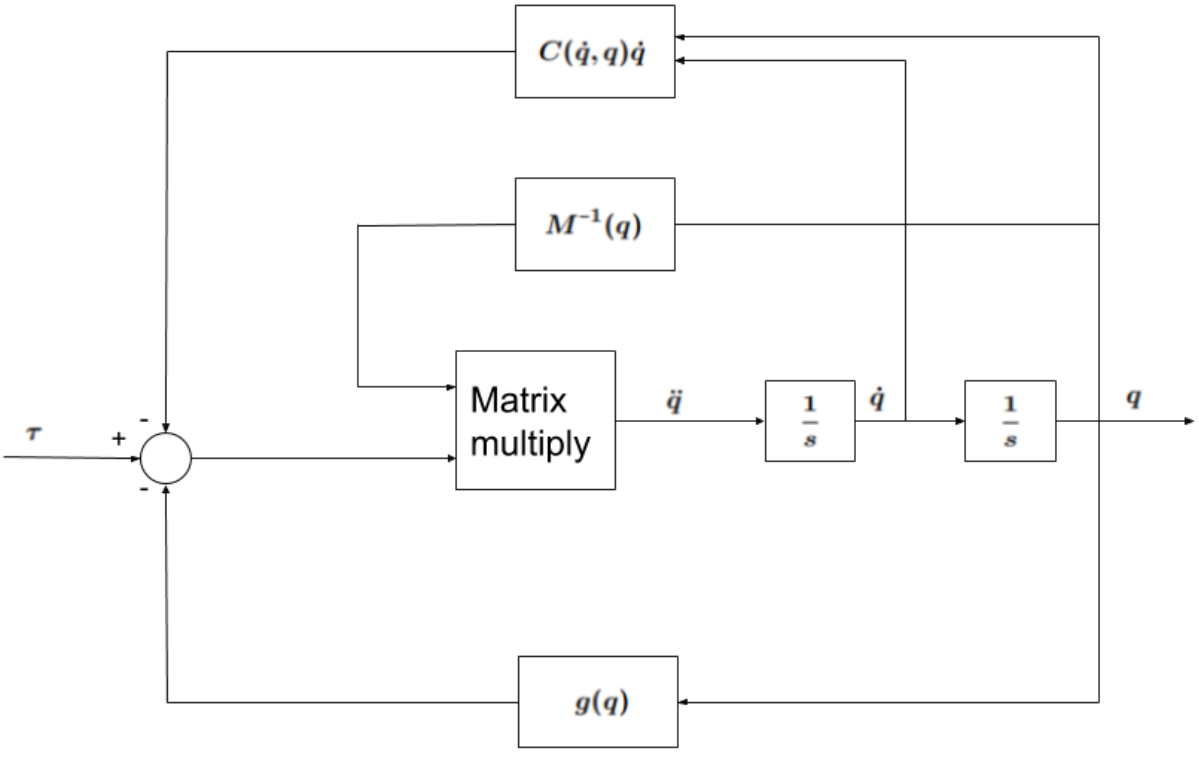



Figure 5.4: Block diagram of the dynamic crane model

This is actually the crane model that is used for modelling in Simulink, and Figure 5.5 shows an overview of how the model is built in Simulink.

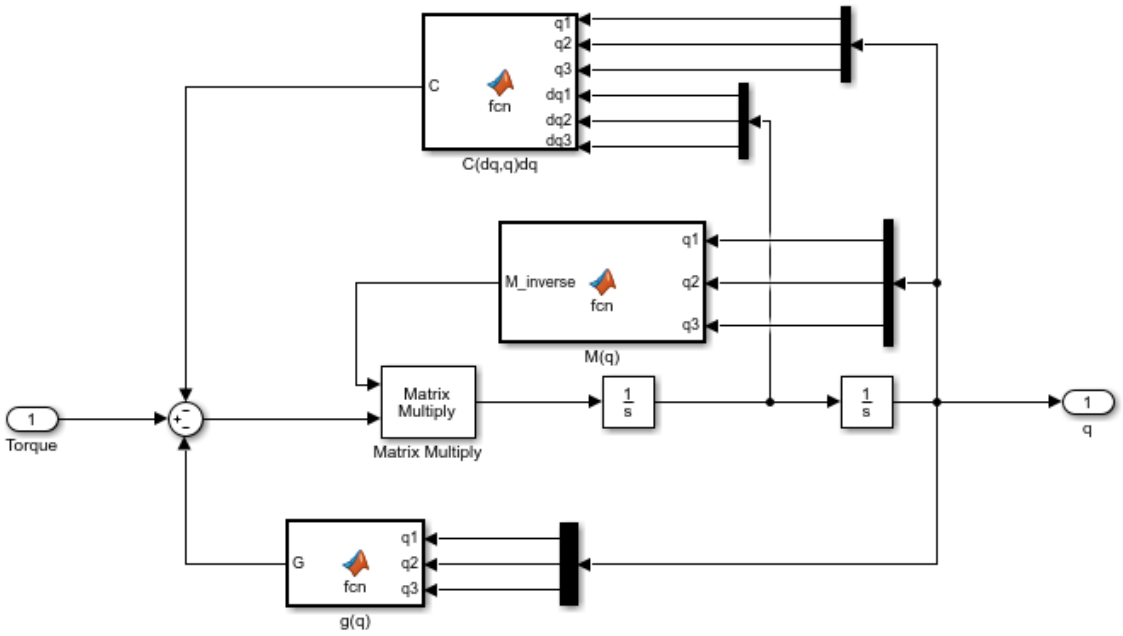



Figure 5.5: Dynamic crane model in Simulink

This model consists of Matlab function blocks of the Inertia matrix, Coriolis and centripetal matrix and Gravity vector. The inverse term of the Inertia matrix is calculated in the Matlab function, and the matrix multiply block execute a matrix multiplication, both in order to fulfill the rearranged equation of motion. $\tau$ is the input torque and the $\frac{1}{s}$ blocks perform an integration of the joint acceleration, and further the joint velocities.

This dynamic crane model will be used in chapter 7, where different control methods will be examined.

# Chapter 6

# Comparison of Matlab and Simulink Crane Model

This chapter includes a comparison of the inverse crane kinematics and dynamics from an analytic model programmed in Matlab and a block diagram model created in Simulink, which can be found in Appendix A. The purpose is to confirm that both models give equal results of trajectory motion, joint motion and joint torque, using the same trajectory generation for both models. The trajectory, which will be used in both models is a circular trajectory in the xy-plane with a simulation time of 30 seconds. The function of this trajectory is to make the end-effector move in a circle in the xy-plane with a radius of 0.5m and a linear velocity of 0.1 m/s.

Before the equations for position, velocity and acceleration are developed, some relations between linear velocity, angular velocity, radius, angle and time are introduced as

$$\omega = \frac{v}{r} \rightarrow v = \omega r \tag{6.1}$$

$$\theta = \omega t \rightarrow \omega = \theta t \tag{6.2}$$

$$\dot{\theta} = \omega \tag{6.3}$$

where

$v$ is the desired linear velocity of the trajectory,

$r$ is the desired radius of the trajectory,

$\omega$ is the angular velocity of the trajectory calculated from the linear velocity and the radius, and

$\theta$ is the angle of the end-effector, calculated from the angular velocity and the time.

Then the equations for end-effector position in x, y and z-direction are

$$x_e = x_0 + rcos(\theta) \tag{6.4}$$

$$y_e = y_0 + rsin(\theta) \tag{6.5}$$

$$z_e = z_0 \tag{6.6}$$

.

By taking the derivative of the position (using the chain rule) the following equations of end-effector velocity in x, y and z direction become

$$\dot{x}_e = cos(\theta) - vsin(\theta) \tag{6.7}$$

$$\dot{y}_e = sin(\theta) + vcos(\theta) \tag{6.8}$$

$$\dot{z}_e = 0 \tag{6.9}$$

and by taking the derivative (also using the chain rule) of the velocities, which is the second derivative of the position, the following equations for end-effector acceleration in x, y and z direction become

$$\ddot{x}_e = -\frac{v}{r}sin(\theta) - \frac{v}{t}sin(\theta) - \frac{v^2}{r}cos(\theta) \tag{6.10}$$

$$\ddot{y}_e = \frac{v}{r}cos(\theta) + \frac{v}{t}cos(\theta) - \frac{v^2}{r}sin(\theta) \tag{6.11}$$

$$\ddot{z}_e = 0 \tag{6.12}$$

Using the above equations for the trajectory combined with the kinematic and dynamic equations, described in chapter 5, the figures below show the resulting end-effector motion, joint motion, cylinder motion and joint torques for the Matlab and Simulink model.

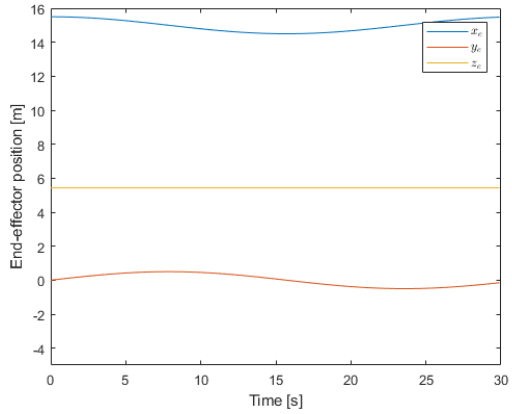



Figure 6.1: End-effector position from Matlab

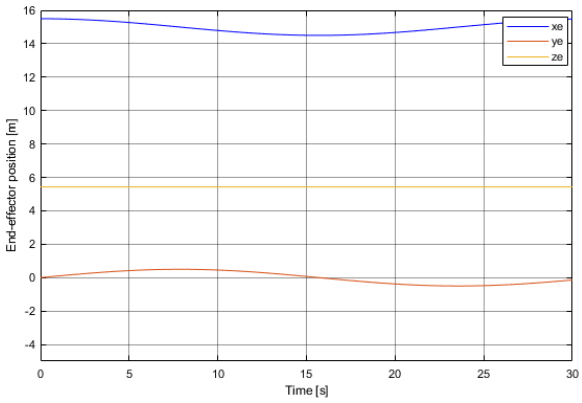



Figure 6.2: End-effector position from Simulink

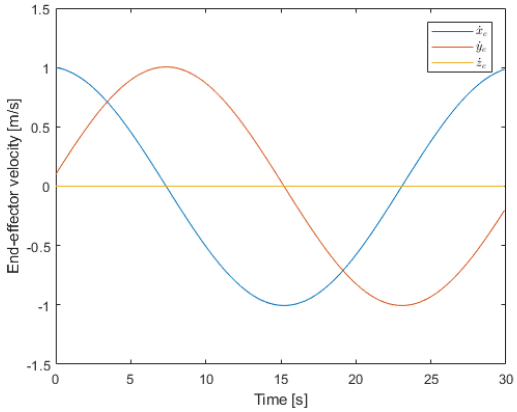



Figure 6.3: End-effector velocity from Matlab

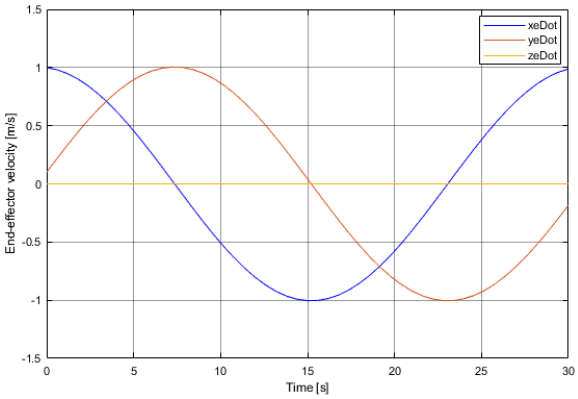



Figure 6.4: End-effector velocity from Simulink

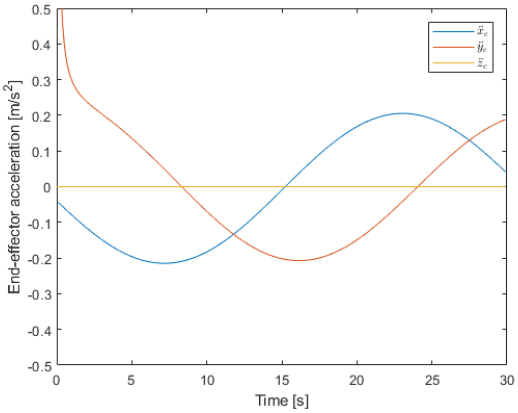



Figure 6.5: End-effector acceleration from Matlab

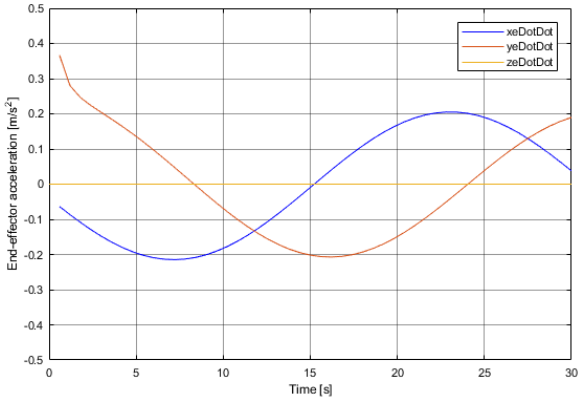



Figure 6.6: End-effector acceleration from Simulink

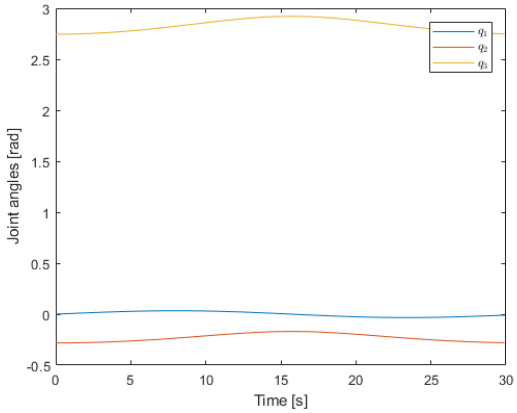



Figure 6.7: Joint angles from Matlab

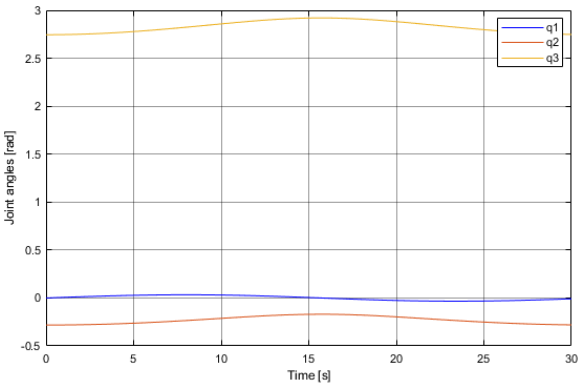



Figure 6.8: Joint angles from Simulink

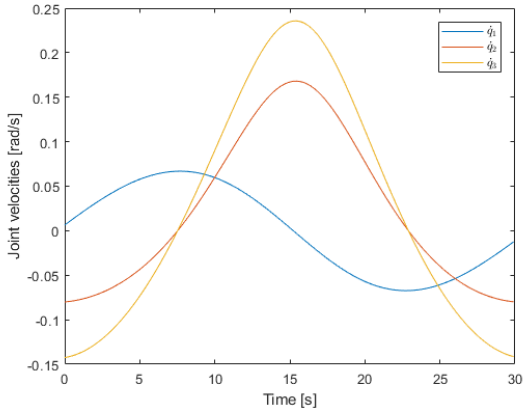



Figure 6.9: Joint velocities from Matlab

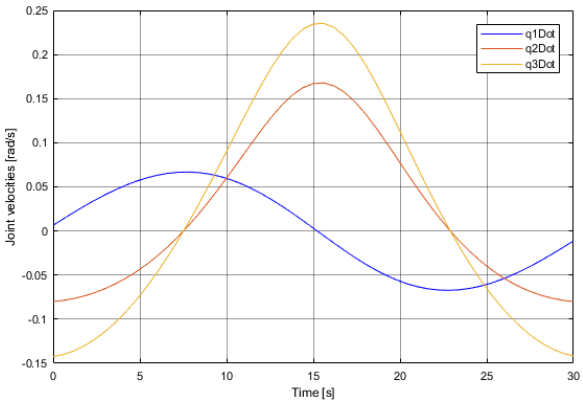



Figure 6.10: Joint velocities from Simulink

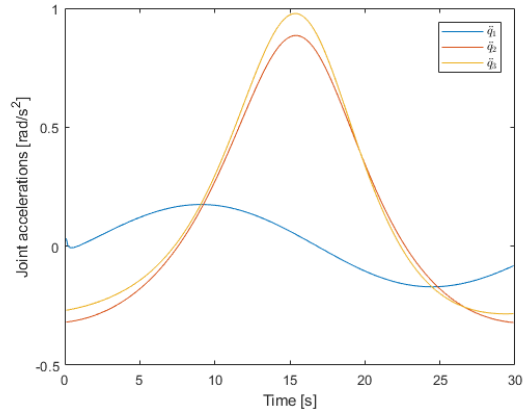



Figure 6.11: Joint accelerations from Matlab

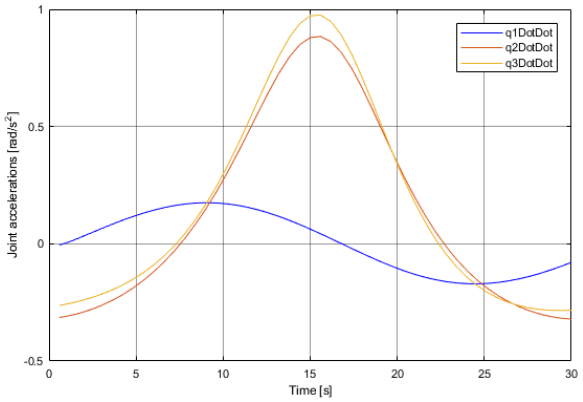



Figure 6.12: Joint accelerations from Simulink

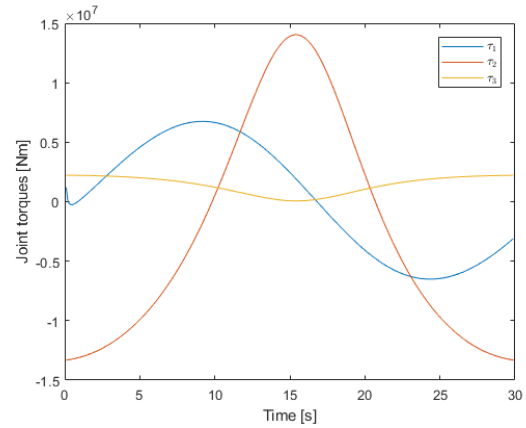



Figure 6.13: Joint torques from Matlab

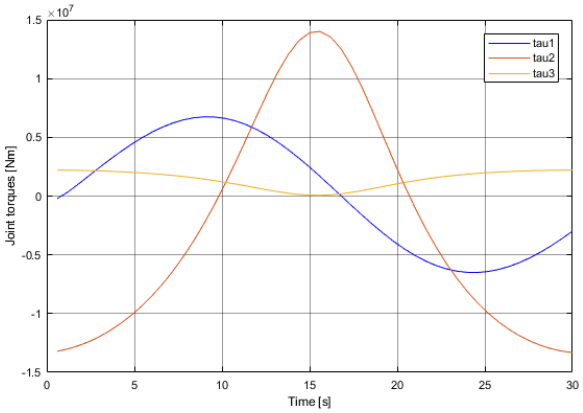



Figure 6.14: Joint torques from Simulink

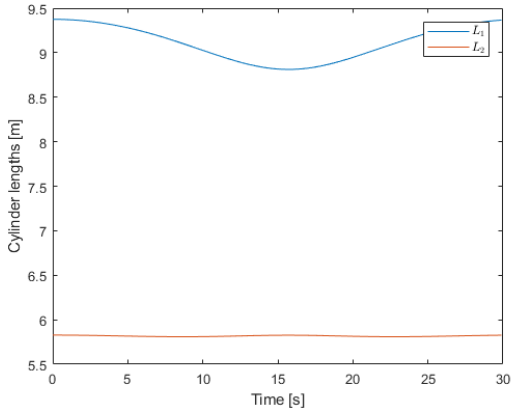



Figure 6.15: Cylinder lengths from Matlab

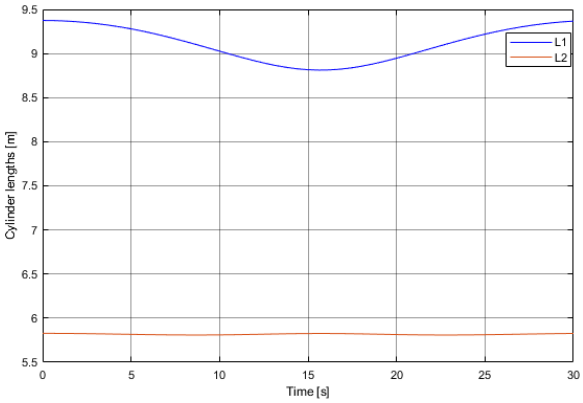



Figure 6.16: Cylinder lengths from Simulink

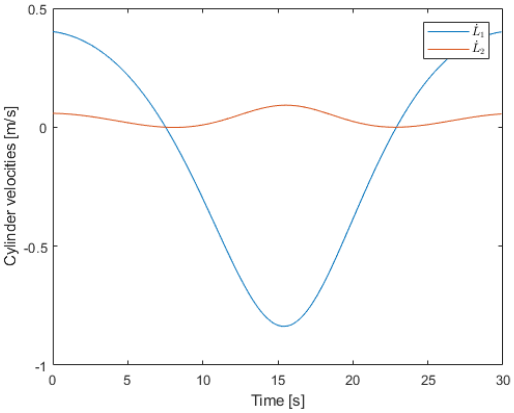



Figure 6.17: Cylinder velocities from Matlab

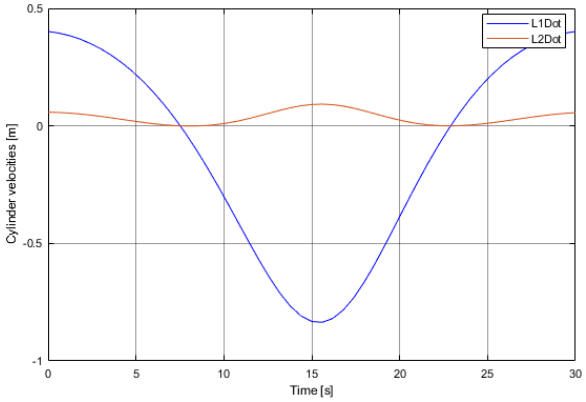



Figure 6.18: Cylinder velocities from Simulink

As seen from the figures above, the two models provide similar results. Both models are based on equations, so an improved verification could have been done using a SimMechanics or a SimulationX model instead of a Simulink model with Matlab function blocks.

# Chapter 7

# Controller Design

This chapter describes several position control designs developed to control the crane in a desired position. This includes direct control of the crane joints and crane end-effector, where the controllers are designed such that the joints and end-effector are expected to follow the movement of a desired reference position or trajectory path. The controller designs that will be presented in this chapter are:

- PID-controller
- PD-controller
- PI-controller
- LQR
- Jacobian inversion method
- Jacobian transpose method

The first three control designs use the dynamic crane model, described in chapter 5.3, as the plant. The same model is used as the plant in the LQR controller design, but now as an estimated state-space model. While the Jacobian inversion method and Jacobian transpose method rely only on the crane kinematics using the Jacobian matrix, described in chapter 5.

First, control designs concerning control of the crane joints and further control designs concerning control of the end-effector are presented, in order to consider the use of each controllers.

## 7.1 Control of Crane Joints

The control task concerning control of crane joints is to get the joint angles to follow a desired joint angle, which in this case is a sine wave with an amplitude of 1, as shown in Figure 7.1. The desired joint angle is used for all three crane joints.

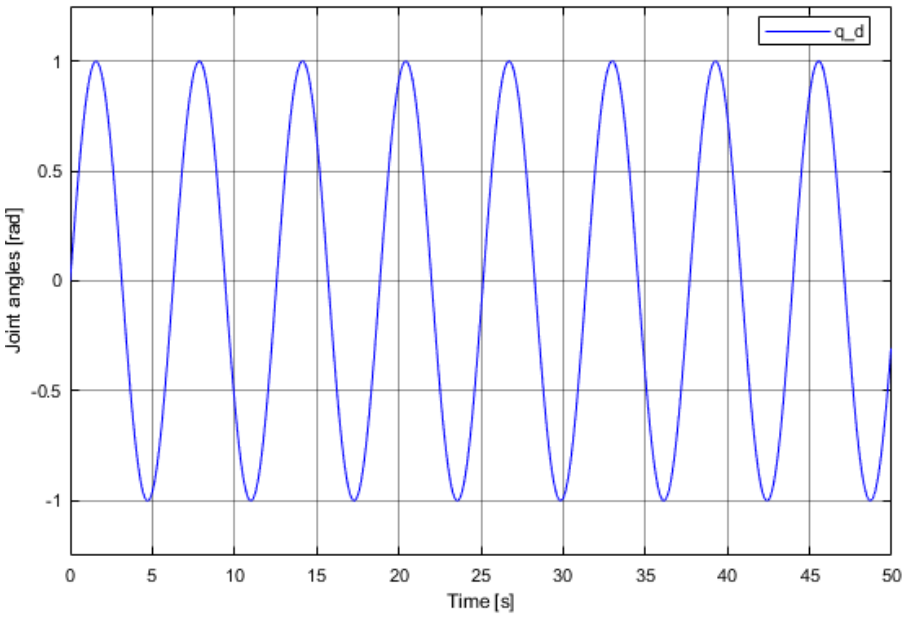



Figure 7.1: Desired joint angles

State of the art controllers such as PID, PI- and PD controllers are developed for this purpose. Because of the heavy crane links that can cause high joint torques, it was also interesting to examine these control designs by implementing gravity compensation to see if this could affect the results. A LQR, which is based on a state-space representation of the dynamic crane model, is also developed for the same purpose.

### 7.1.1 PID-controller

To get the measured joint angles to follow the desired joint angles (described in chapter 7.1) with as small error as possible, the PID-controller design consists of a PID-controller that continuously calculates an error value as the difference between desired and measured joint angles and applies a correction based on proportional, integral and derivative terms. The proportional gain $K_p$ has the ability to reduce the error and can also cause the closed-loop system to react faster, but also to overshoot more. The derivative gain $K_d$ has the ability of the controller to "anticipate" error, and then reduce the overshoot. The integral gain $K_i$ can help reducing the error. If there is a persistent steady error, $K_i$ can drive the error down. Because of the heavy crane links, compensating for gravity may affect the system response, thus the PID-controller design is also developed by adding the gravity vector from the dynamic system to the input torque.

Figure 7.2 shows how the PID-controller is developed with the task of tracking the desired joint angles. The controller parameters $K_p$, $K_i$ and $K_d$ were found from testing several values and combinations. Figure 7.3 shows how the same controller is built when compensating for gravity.

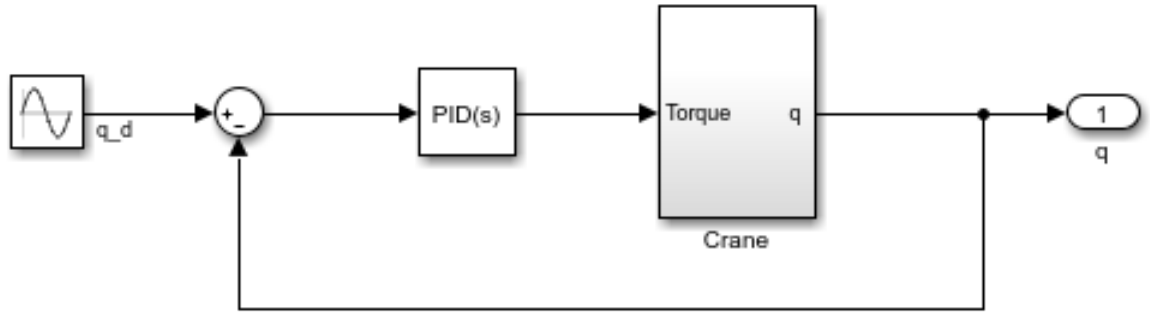



Figure 7.2: Control of joint angles using PID-controller

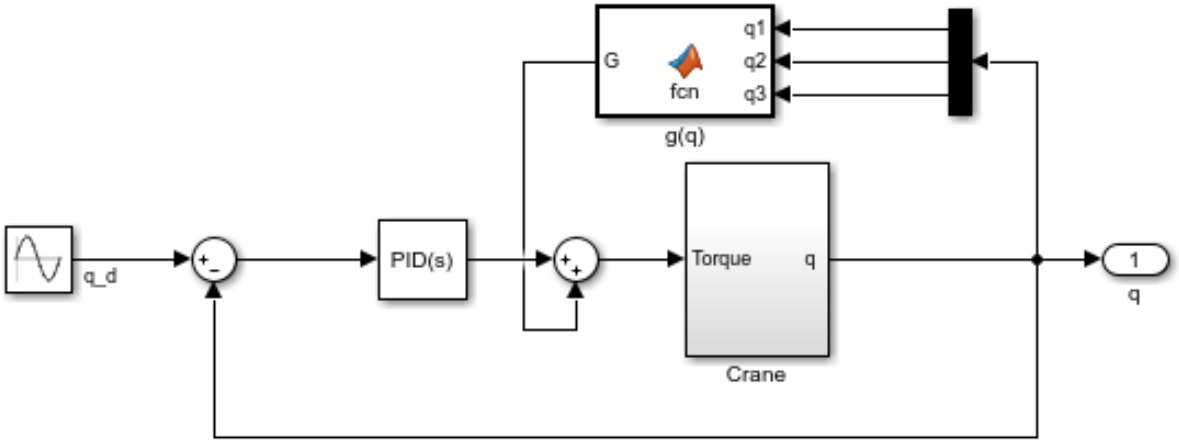



Figure 7.3: Control of joint angles using PID-controller with gravity compensation

### 7.1.2 PD-controller

To get the measured joint angles to follow the desired joint angles (described in chapter 7.1) with as small error as possible, the PD-controller design consists of a PD-controller that continuously calculates an error value as the difference between desired and measured joint angles and applies a correction based on proportional and derivative terms. Even if the integral gain can help reducing the error, it can also make the system more oscillatory. Therefore, it is expedient to see how the system will react without the integral term. Because of the heavy crane links, compensating for gravity may affect the system response, thus the gravity vector from the dynamic system is added to the input torque.

Figure 7.4 shows how the PD-controller is developed with the task of tracking the desired joint angles. The controller parameters $K_p$ and $K_d$ were found from testing several values and combinations. Figure 7.5 shows how the same controller is built when compensating for gravity.

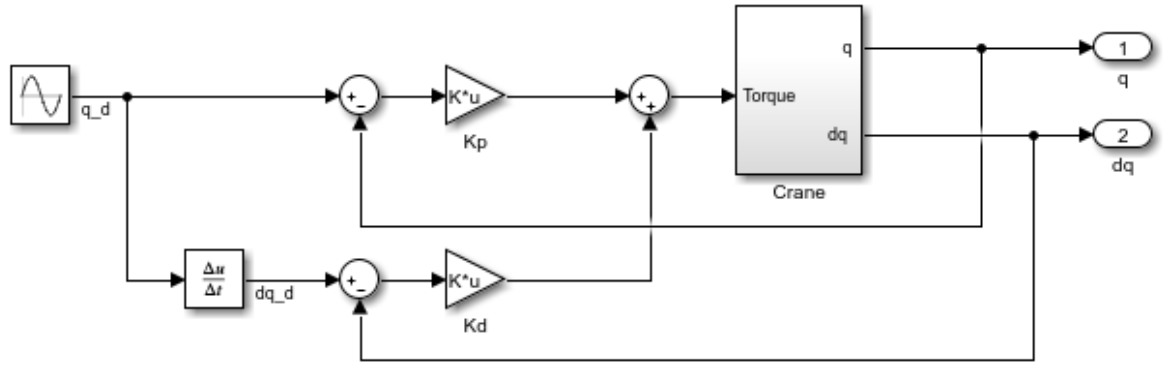



Figure 7.4: Control of joint angles using PD-controller

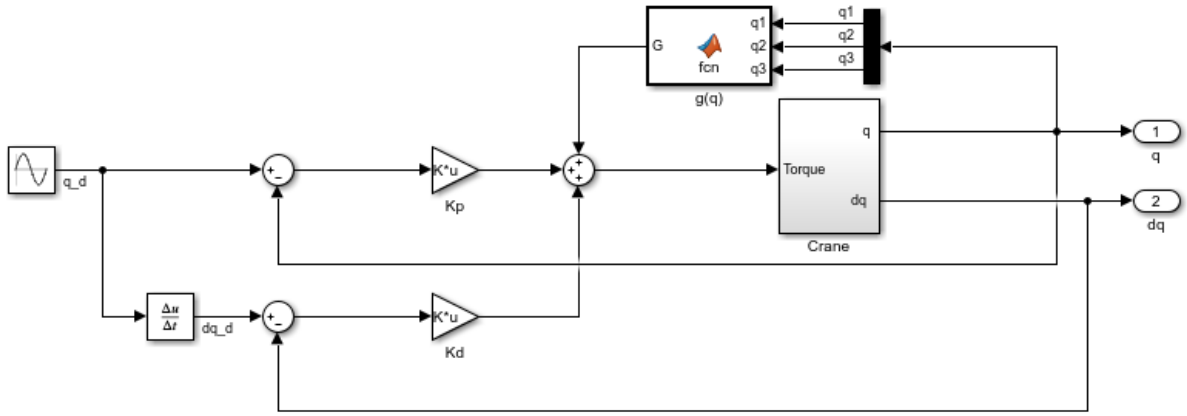



Figure 7.5: Control of joint angles using PD-controller with gravity compensation

### 7.1.3 PI-controller

To get the measured joint angles to follow the desired joint angles (described in chapter 7.1) with as small error as possible, the PI-controller design consists of a PI-controller that continuously calculates an error value as the difference between desired and measured joint angles and applies a correction based on proportional and integral terms. Even if the derivative gain $K_d$ helps reducing the overshoot, it has no effect on the steady-state error. Therefore, it is expedient to examine how the system will react without the derivative gain. Because of the heavy crane links, compensating for gravity may affect the system response, thus the PI-controller design is also developed by adding the gravity vector from the dynamic system to the input torque.

Figure 7.6 shows how the PI-controller is developed with the task of tracking the desired joint angles. The controller parameters $K_p$ and $K_i$ were found from testing several values and combinations. Figure 7.7 shows how the same controller is built when compensating for gravity.

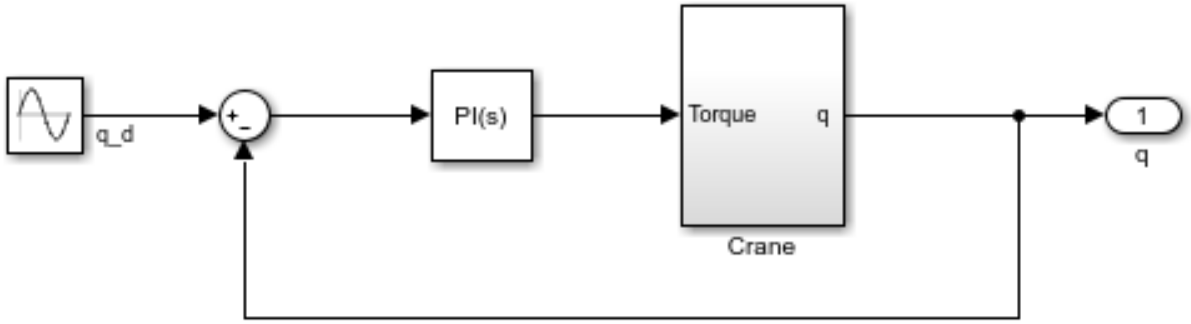



Figure 7.6: Control of joint angles using PI-controller

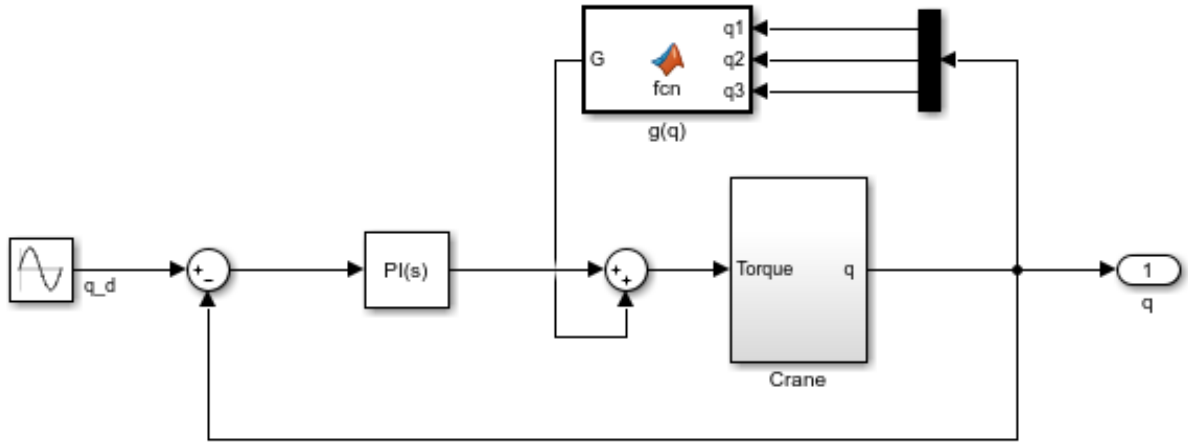



Figure 7.7: Control of joint angles using PI-controller with gravity compensation

### 7.1.4 LQR

To get the measured joint angles to follow the desired joint angles (described in chapter 7.1) with as small error as possible, the LQR control design consists of an estimated state-space model of the dynamic crane with the joint angles as the outputs, a state feedback gain matrix that will minimize the cost function and may reduce the error between desired and measured joint angles and a pre-filter to further reduce the error.

Figure 7.8-7.10 show the structure of a LQR for each of the crane joints with the task of tracking the desired joint angles. These models are run in parallel with the Matlab model, shown in Appendix B. The plant for each of the models is an estimated model of the crane with joint angles as the output and are found from the dynamic crane model in Simulink, which can be seen as a multiple-input and multiple-output system. A sine wave is set as the input for all three joints and crane joint 1, 2 and 3 are the outputs. Further, "System identification toolbox" in Matlab is used to find an estimated state-space model for each input with the associated output. All three state-space models had an estimation fit to the dynamic crane model of nearly 100 percent. Therefore, these models are used as the plant instead of the dynamic crane model, and the crane joints can be controlled separately. In addition, the LQR design of joint 2 includes a

feedback gain to the system. This is done to improve the results and minimize the error between the desired and measured joint angle.

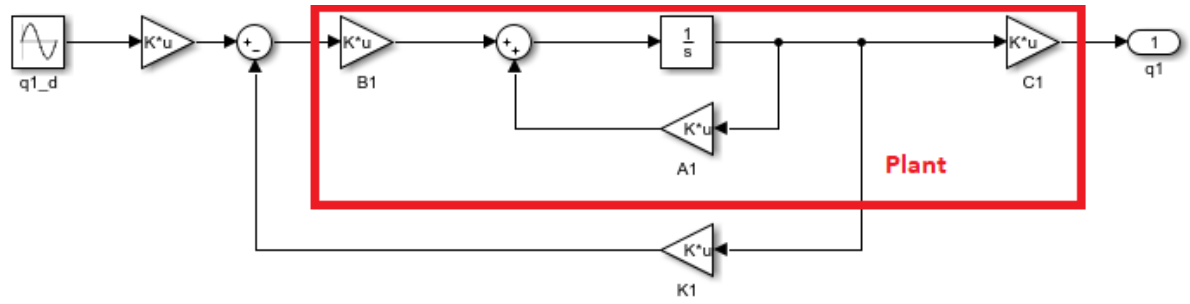



Figure 7.8: Control of joint angle 1 using LQR

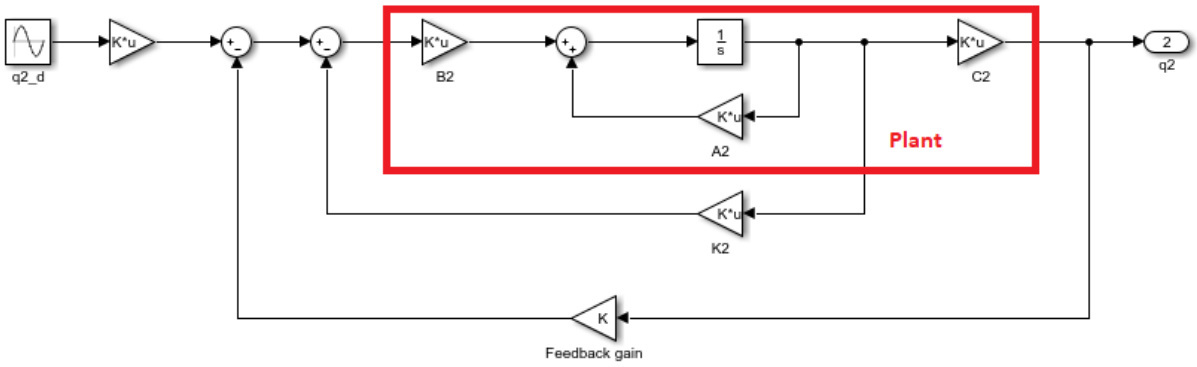



Figure 7.9: Control of joint angle 3 using LQR

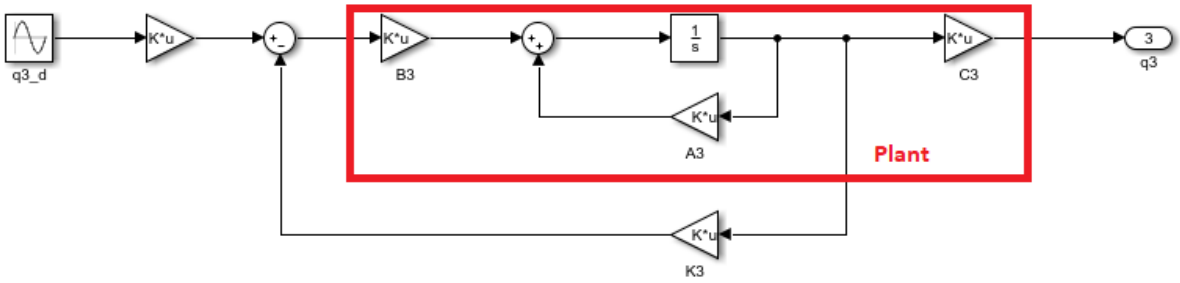



Figure 7.10: Control of joint angle 3 using LQR

When the estimated state-space models were found, a proper state feedback gain for each system were calculated in Matlab using the following equations

$$K_1 = lqr(A_1, B_1, Q_1, R_1) \tag{7.1}$$

$$K_2 = lqr(A_2, B_2, Q_2, R_2) \tag{7.2}$$

$$K_1 = lqr(A_3, B_3, Q_3, R_3) \tag{7.3}$$

where Equation (7.1) represents the gain matrix for control of joint angle 1, Equation (7.2) represents the gain matrix for control of joint angle 2 and Equation (7.3) represents the gain matrix for control of joint angle 3.

To improve the results and reduce the error between desired and measured joint angles a pre-filter is added to controllers. The pre-filter values are calculated with the following equations

$$V_1 = (C_1(B_1K_1 - A_1)^{-1})B_1)^{-1} \tag{7.4}$$

$$V_2 = (C_2(B_2K_2 - A_2)^{-1})B_2)^{-1} \tag{7.5}$$

$$V_3 = (C_3(B_3K_3 - A_3)^{-1})B_3)^{-1} \tag{7.6}$$

where Equation (7.4) represents the pre-filter for control of joint angle 1, Equation (7.5) represents the pre-filter for control of joint angle 2 and Equation (7.6) represents the pre-filter for control of joint angle 3.

## 7.2 Control of Crane End-effector

The control task concerning control of crane end-effector is to get the crane end-effector to follow a desired trajectory movement in vertical direction. The desired trajectory is to move the end-effector from an initial point to another desired point in z-direction, with a velocity of 0.1 m/s. The initial location is set to 5.432 m, which is the z-coordinate when the crane arm is fully extended, and the desired location is set to 1 m. After the end-effector has reached the desired location, the aim is to keep the end-effector steady at this point. Figure 7.11 shows the desired trajectory of the crane end-effector.

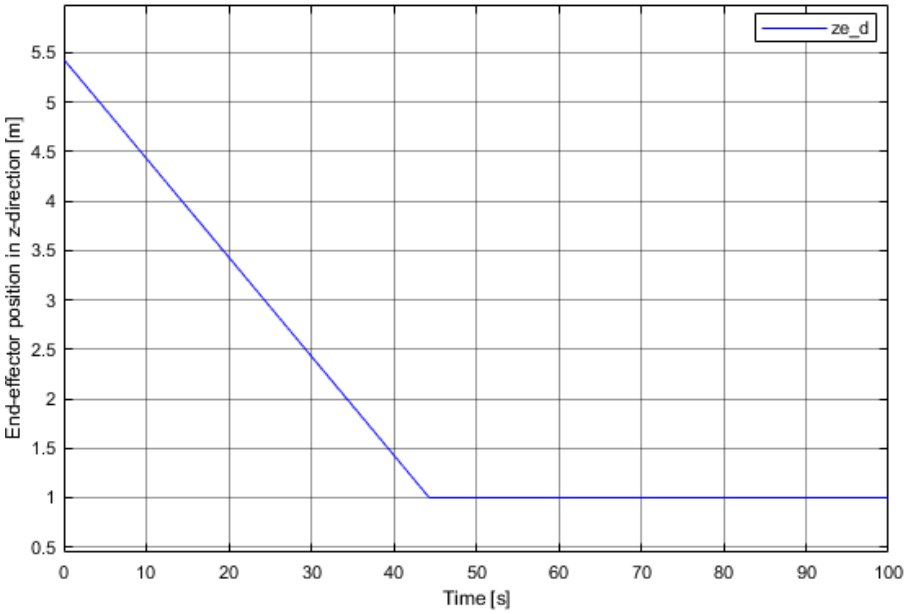



Figure 7.11: Desired position in z-direction

The trajectory can be described as a linear movement in z-direction with the following equation

$$z_{ed} = z_{e0} + mt \tag{7.7}$$

where

$z_{ed}$ is the desired movement of the end-effector in z-direction given in meters,

$z_{e0}$ is the initial position of the end-effector in z-direction given in meters,

$t$ is the time given in seconds, and

$m$ is the slope of the movement.

The equation of the slope can be written as

$$m = \frac{z_{ef} - z_{e0}}{t_f - t_0} \tag{7.8}$$

where

$z_{ef}$ is the desired final end-effector location given in meter,

$t_f$ is the final time given in seconds, and

$t_0$ is the inital time given in seconds.

The final time can be calculated from the difference in end-effector position divided with the difference in end-effector velocity as follows

$$t_f = \frac{z_{ef} - z_{e0}}{v_{ef} - v_{e0}} \tag{7.9}$$

where

$v_{ef}$ is the desired end-effector velocity in z-direction (in negative direction) given in meter per seconds, and

$v_{ef}$ is the initial velocity of the end-effector in z-direction given in meter per seconds.

The controllers used for this purpose are first the state-of-art controllers such as PID-, PI- and PD-controllers, where implementing gravity compensation seemed to be necessary to avoid singularity in the solution and further track the desired end-effector position. Ziegler-Nichols closed-loop tuning method to find controller parameters for these controllers is also explained. Further, a LQR is developed for the same purpose. Eventually, the inverse kinematics control method called the Jacobian inversion and Jacobian transpose method are developed.

### 7.2.1 Ziegler-Nichols Closed-loop Tuning

To find the controller parameters for the PID-, PI- and PD-controllers, Ziegler Nichols closed-loop tuning method is used. A controller is connected to the system using only the proportional gain. The value of this gain is increased until the system starts to oscillate, as shown in Figure 7.12. Further, the ultimate period $P_u$ is found using "Peak finder" in Simulink. This is shown in Figure 7.13. Then the equations in Table 3.4 are used to find the controller parameters for all three controllers.

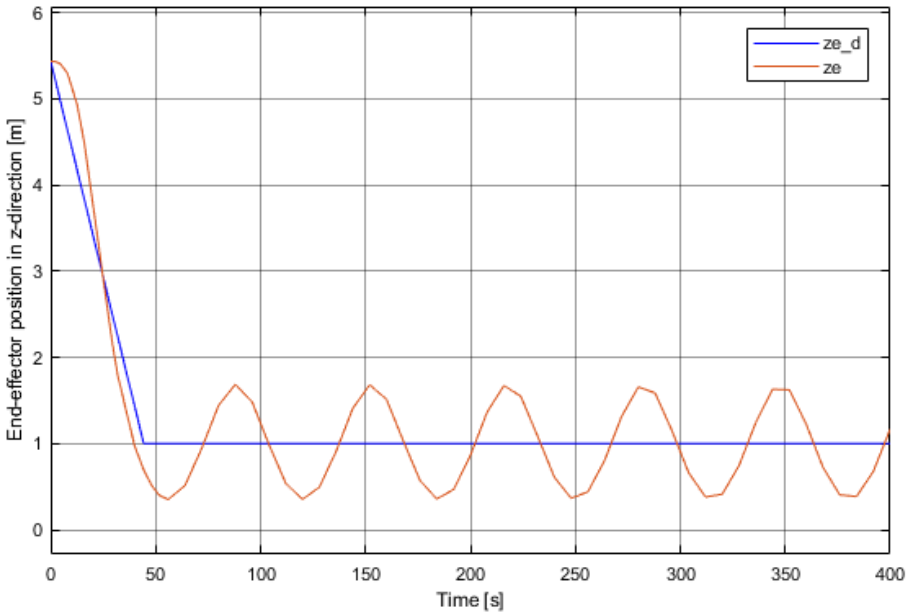



Figure 7.12: Procedure to find the ultimate gain $K_u$

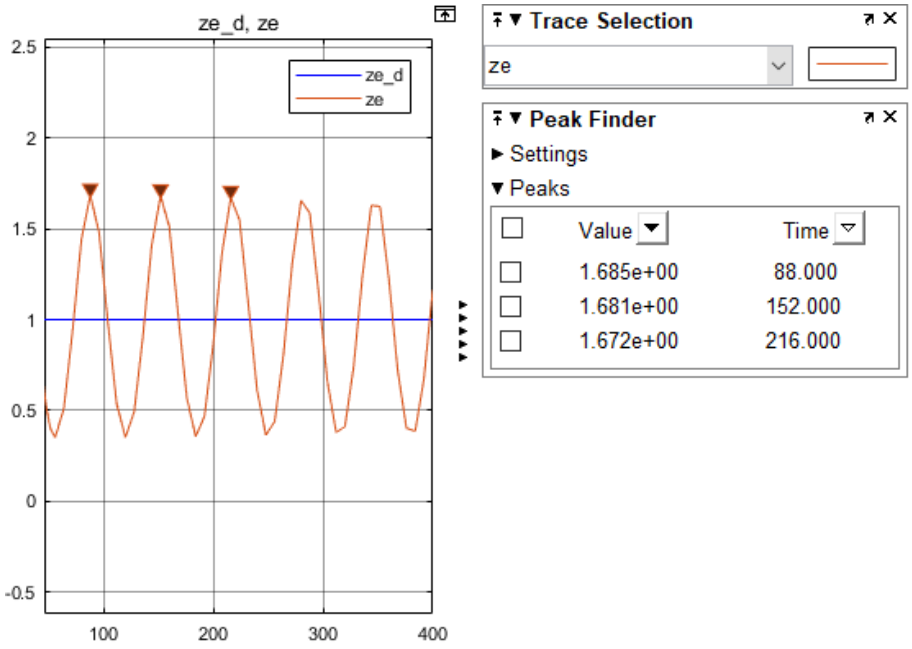



Figure 7.13: Using peak finder to determine the ultimate period Pu

### 7.2.2 PID-controller

To get the measured end-effector position in z-direction to follow the desired end-effector position in z-direction (described in chapter 7.2) with as small error as possible, the PID-controller design consists of a PID-controller that continuously calculates an error value as the difference between desired and measured position and applies a correction based on proportional, integral and derivative terms. The proportional gain $K_p$ has the ability to reduce the error and can also cause the closed-loop system to react faster but also to overshoot more. The derivative gain $K_d$ has the ability of the controller to "anticipate" error and then reduce the overshoot. The integral gain $K_i$ can help reducing the error. If there is a persistent steady error, $K_i$ can drive the error down. Because of the heavy crane links, compensating for gravity may affect the

system response, thus the PID-controller design is developed by adding the gravity vector from the dynamic system to the input torque. This also turned out to be necessary for simulating the system in Simulink.

Figure 7.14 shows how the PID-controller design is developed for the end-effector with the task of tracking the desired trajectory.

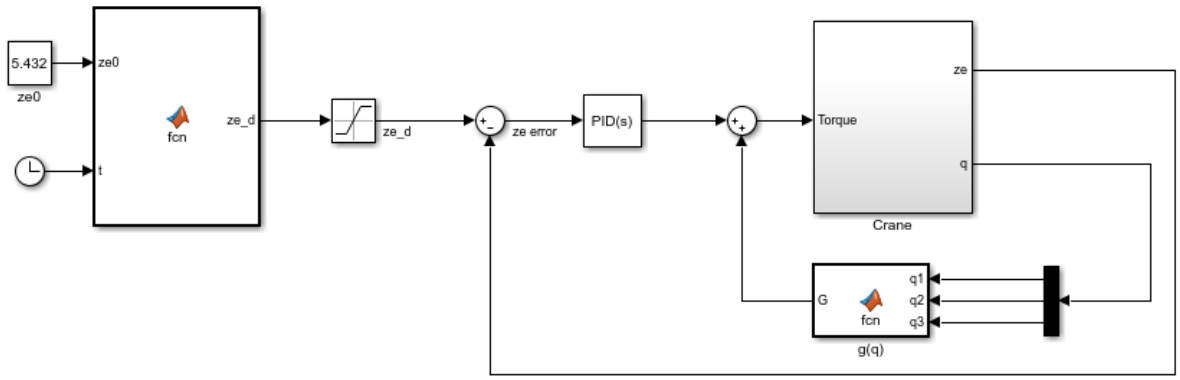



Figure 7.14: Control of end-effector position using PID-controller with gravity compensation

### 7.2.3 PD-controller

To get the measured end-effector position in z-direction to follow the desired end-effector position (described in chapter 7.2) with as small error as possible the PD-controller design consists of a PD-controller that continuously calculates an error value as the difference between desired and measured position and applies a correction based on proportional and derivative terms. Even if the integral gain can help reducing the error, it can also make the system more oscillatory. Therefore, it is expedient to see how the system will react without the integral term. Because how the heavy crane links, compensating for gravity may affect the system response, thus the gravity vector from the dynamic system is added to the input torque. This also turned out to be necessary for simulating the system in Simulink.

Figure 7.15 shows how the PD-controller design is built for the end-effector with the task of tracking the desired trajectory.

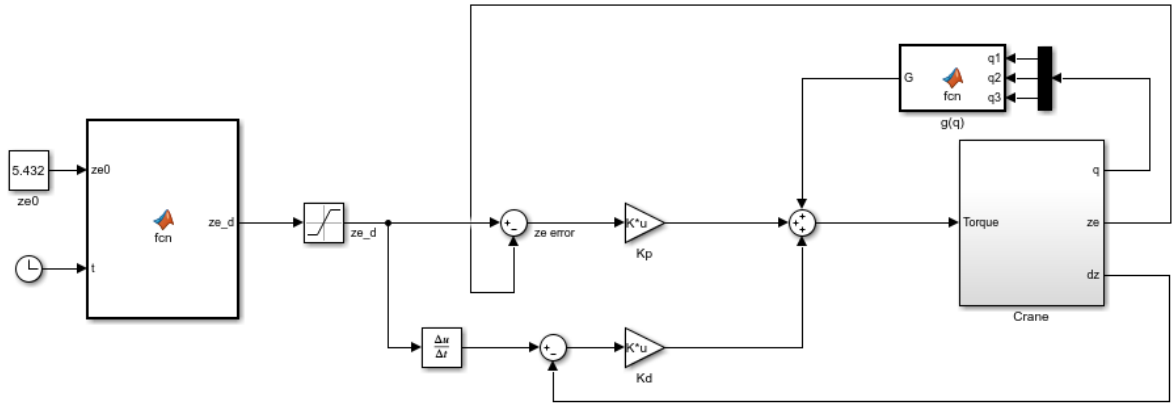



Figure 7.15: Control of end-effector position using PD-controller with gravity compensation

### 7.2.4 PI-controller

To get the measured end-effector position in z-direction to follow the desired end-effector position (described in chapter 7.2) with as small error as possible, the PI-controller design consists of a PI-controller that continuously calculates an error value as the difference between desired and measured position and applies a correction based on proportional and integral terms. Even if the derivative gain $K_d$ helps reducing the overshoot it has no effect the steady-state error. Therefore, it is expedient to examine how the system will react without the derivative gain. Because of the heavy crane links, compensating for gravity may affect the system response, thus the PI-controller design is developed by adding the gravity vector from the dynamic system to the input torque. This also turned out to be necessary for simulating the system in Simulink.

Figure 7.16 shows how the PI-controller design is developed for the end-effector with the task of tracking the desired trajectory.

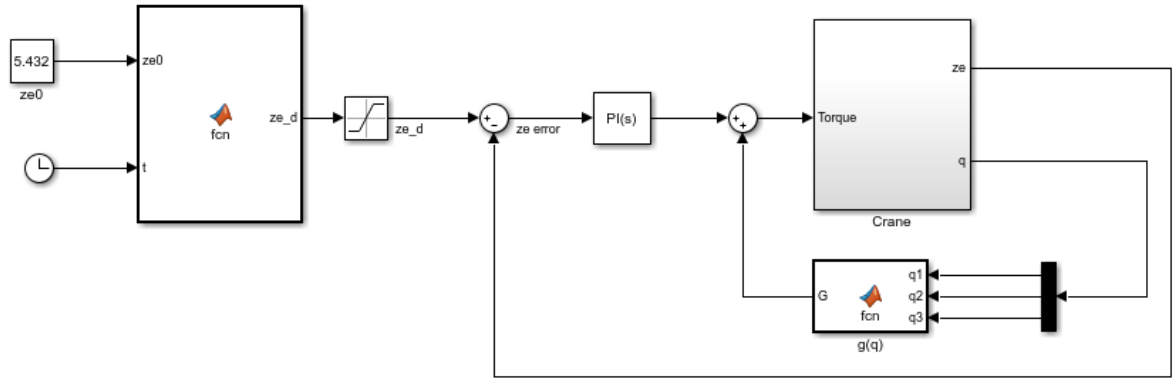



Figure 7.16: Control of end-effector position using PI-controller with gravity compensation

### 7.2.5 LQR

To get the measured end-effector position in z-direction to follow the desired end-effector position in z-direction (described in chapter 7.2) with as small error as possible, the LQR control design consist of an estimated state-space model of the dynamic crane with the end-effector position in z-direction as the output, a state feedback gain matrix that will minimize the cost function and may reduce the error between desired and measured position, and a pre-filter to further reduce the error.

Figure 7.17 shows the structure of a LQR controller design with the task of tracking the desired end-effector position. This model is run in parallel with the Matlab model, shown in Appendix C. The plant is an estimated model of the crane model with end-effector position in z-direction as the output. This model is, like the joint models, found from the dynamic model of the crane in Simulink. The only difference is that the system is now seen as a single-input and single-output system. A step function is set as the input and end-effector position in z-direction is the output. Using "System Identification toolbox" in Matlab, the estimated state-space model had an estimation fit to the dynamic crane model of nearly 100 percent. Therefore, this model is used as the plant instead of the dynamic model.

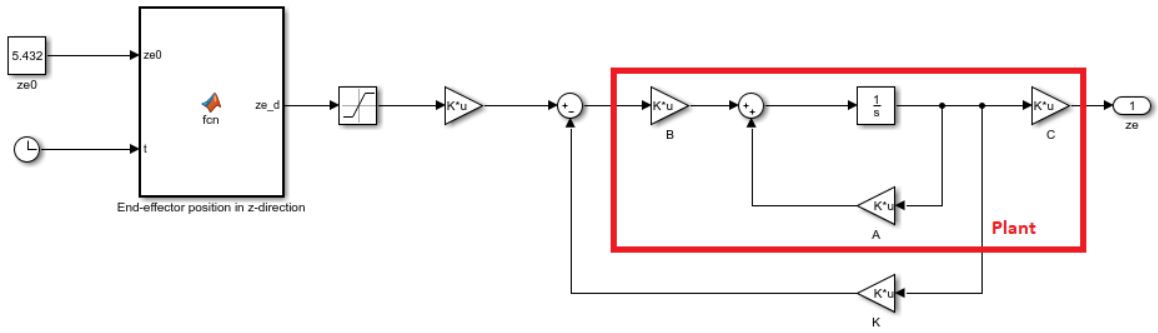



Figure 7.17: Control of end-effector position using LQR

When the estimated state-space models were found, a proper state feedback gain for the system was calculated in Matlab using the following equation

$$K = lqr(A, B, Q, R) \tag{7.10}$$

To improve the results and reduce the error between the desired and measured end-effector position a pre-filter is added to controller. The pre-filter value is calculated with the following equation

$$V = (C(BK - A)^{-1})B)^{-1} \tag{7.11}$$

### 7.2.6 Jacobian Inversion Method

The Jacobian inversion method is a control method that relies only on the crane kinematics and not the dynamics, where the purpose is to minimize the difference between desired and current position of the end-effector. It is an iterative inverse kinematics method, which means that it solves the kinematic equations using a sequence of steps. This provides a better solution for the joint angles, as an algebraic solution works only for a restricted class of cases and can be used for a 2DOF crane. Since this concerns a 3DOF crane, it is necessary to utilize an iterative method using the Jacobian matrix instead of algebraic equations. One concern with this method is that using the inverse Jacobian matrix may not lead to one solution, but an infinite number of solutions and singularities usually occur.

Figure 7.18 shows how the controller design is developed using Jacobian inversion method. The design consists of a desired end-effector position, which will be compared to the current end-effector position. This is done by calculating the error between desired and current position. Further, the position error is multiplied with the inverse Jacobian matrix to calculate the joint velocities. The desired trajectory is the one used for the other controller designs described above, but to use the Jacobian, the inverse of the Jacobian and the forward kinematics, it was necessary to define the crane end-effector position, not only in z-direction, but also in x- and y-direction. This is simply done by calculating how the end-effector will move in x- and y-direction based on the position in z-direction.

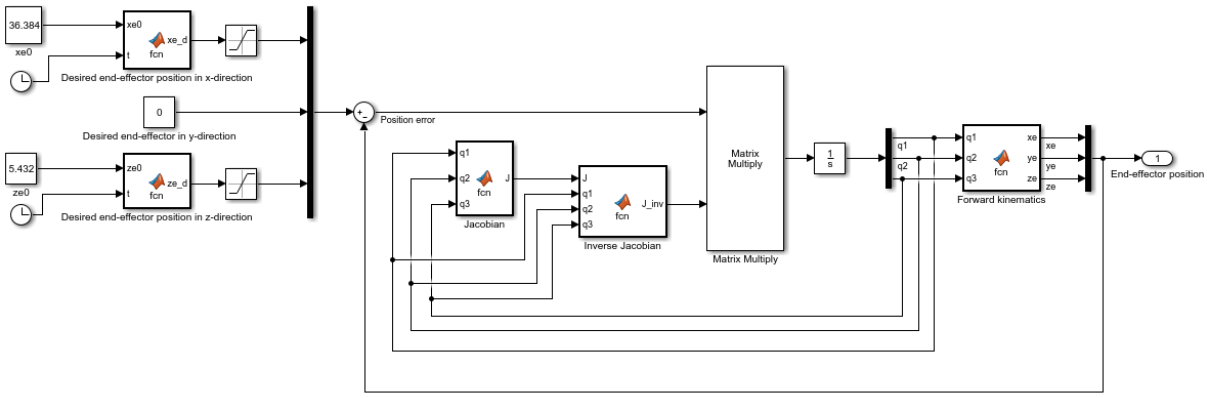



Figure 7.18: Control of end-effector position using Jacobian inversion method

### 7.2.7 Jacobian Transpose Method

The Jacobian transpose method is also an iterative kinematics method. It differs from the Jacobian inversion method as it uses the transpose of the Jacobian matrix instead of the inverse. The purpose is the same for the Jacobian transpose method as for the Jacobian inversion method; to minimize the difference between desired and current position of the end-effector, but using the transpose of the Jacobian, instead of the inverse, removes the singularity problems significantly.

Figure 7.19 shows how the controller design is developed using the Jacobian transpose method. This design consists of a desired end-effector position, the transpose of the Jacobian matrix and forward kinematics that calculates the end-effector position. The desired trajectory is the one used for the Jacobian inversion method, but here the position error can be seen as the generated force, which is proportional with the joint velocities. Therefore, the transpose Jacobian can be used to calculate the end-effector velocities from the generated force.

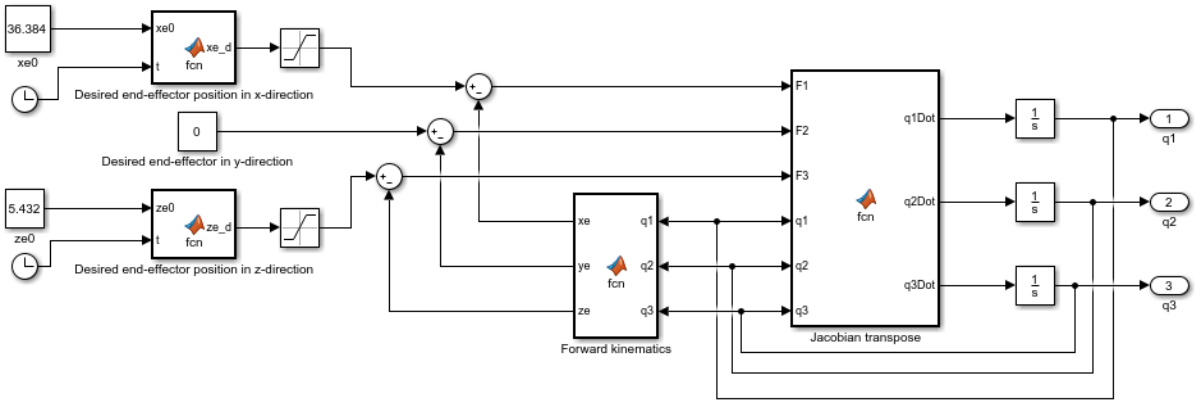



Figure 7.19: Control of end-effector position using Jacobian transpose method

# Chapter 8

# Simulation Results and Discussion

In this chapter simulation results from the control designs will be presented. This includes results from the control designs concerning control of joints and control designs concerning control of end-effector position in vertical directing. Both with the task of tracking a desired position.

## 8.1 Control of Crane Joints

First, results from simulation of the control designs concerning control of crane joints will be presented. This consist of results from controller designs as follows

- PID-controller with and without gravity compensation
- PD-controller with and without gravity compensation
- PI-controller with and without gravity compensation
- LQR

The measured joint angles will be compared to the desired joint angles to examine the ability each controller design has to track the desired joint angles. When the control task concerning control of crane joints is to get the joint angles to follow a desired joint angle, there will occur a certain error as the difference between the desired and measured joint angle. This error will be presented in form of a graph with respect to time and with a maximal value. To fulfill the control task, controller parameters need to be found or calculated, which will be presented as well.

### 8.1.1 PID-controller

Results from simulation of the PID-controller design include measured joint angles versus desired joint angles and the error between desired and measured joint angles. Controller parameters for the PID-controller design are found from testing several values. The best results were obtained when $K_p$, $K_i$ and $K_d$ where increased to their limits, which eventually resulted in very high gains.

Figure 8.1 shows the desired and measured joint angles and Figure 8.2 shows the error as the difference between desired and measured joint angles.

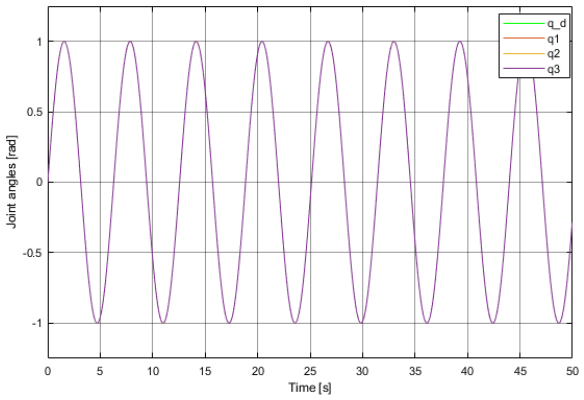



Figure 8.1: Measured joint angles versus desired joint angles using PID-controller

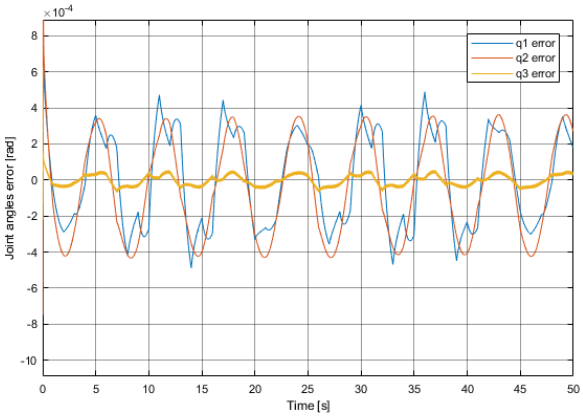



Figure 8.2: Error between desired and measured joint angles using PID-controller

Maximal error between desired and measured joint angles is found from Figure 8.2 and the values for each joint variable are:

- $q_1$ error= $2.775 \cdot 10^{-3}[rad]$ at time $t = 4.438 \cdot 10^{-3}[s]$
- $q_2$ error=$3.327 \cdot 10^{-3}[rad]$ at time $t = 4.717 \cdot 10^{-3}[s]$
- $q_3$ error=$6.268 \cdot 10^{-4}[rad]$ at time $t = 3.831 \cdot 10^{-3}[s]$

### 8.1.2 PID-controller with Gravity Compensation

Results from simulation of the PID-controller with gravity compensation design include measured joint angles versus desired joint angles and the error between desired and measured joint angles.

Using the same controller parameters as for the PID-controller design, Figure 8.3 shows the desired and measured joint angles and Figure 8.4 shows the error as the difference between desired and measured joint angles, when adding gravity compensation.

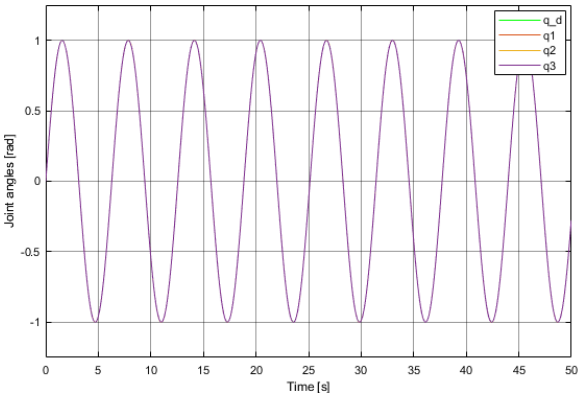



Figure 8.3: Measured joint angles versus desired joint angles using PID-controller with gravity compensation

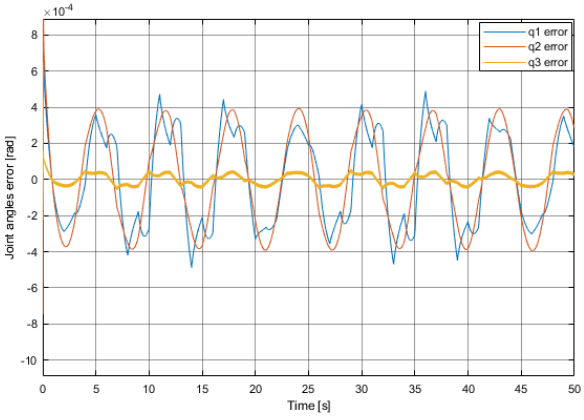



Figure 8.4: Error between desired and measured joint angles using PID-controller with gravity compensation

Maximal error between desired and measured joint angles is found from Figure 8.4 and the values for each joint variable are:

- $q_1$ error= $2.776 \cdot 10^{-3}[rad]$ at time $t = 4.482 \cdot 10^{-3}[s]$

- $q_2$ error=$3.331 \cdot 10^{-3}[rad]$ at time $t = 4.765 \cdot 10^{-3}[s]$
- $q_3$ error=$6.464 \cdot 10^{-4}[rad]$ at time $t = 4.765 \cdot 10^{-3}[s]$

### 8.1.3 PD-controller

Results from simulation of the PD-controller design include measured joint angles versus desired joint angles and the error between desired and measured joint angles. Controller parameters for the PD-controller design are also found from testing several values. The best results were obtained when $K_p$ and $K_d$ where increased to their limits, which eventually resulted in very high gains.

Figure 8.5 shows the desired and measured joint angles and Figure 8.6 shows the error as the difference between desired and measured joint angles.

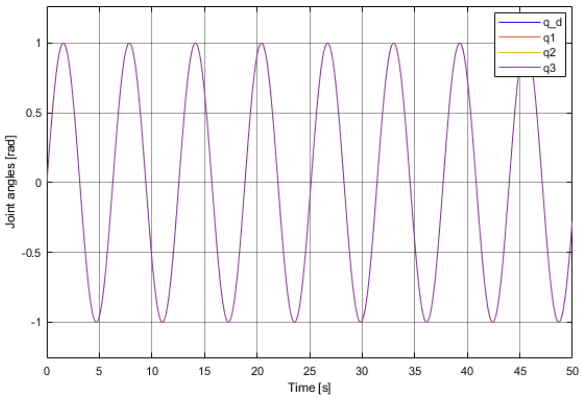



Figure 8.5: Measured joint angles versus desired joint angles using PD-controller

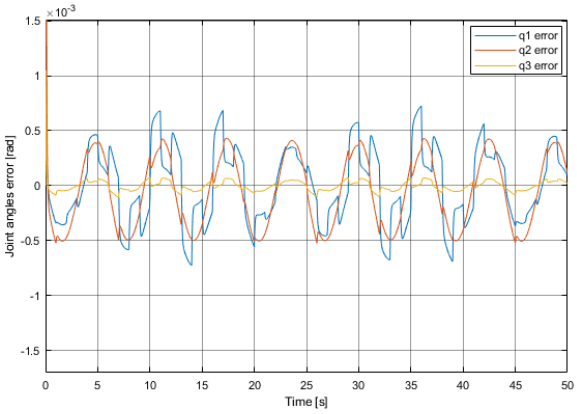



Figure 8.6: Error between desired and measured joint angles using PD-controller

Maximal error between desired and measured joint angles is found from Figure 8.6 and the values for each joint variable are:

- $q_1$ error= $5.725 \cdot 10^{-3}[rad]$ at time $t = 0.018[s]$
- $q_2$ error=$7.079 \cdot 10^{-3}[rad]$ at time $t = 0.019[s]$
- $q_3$ error=$1.049 \cdot 10^{-3}[rad]$ at time $t = 0.019[s]$

### 8.1.4 PD-controller with Gravity Compensation

Results from simulation of the PD-controller with gravity compensation design include measured joint angles versus desired joint angles and the error between desired and measured joint angles.

Using the same controller parameters as for the PD-controller design, Figure 8.7 shows the desired and measured joint angles and Figure 8.8 shows the error as the difference between desired and measured joint angles, when adding gravity compensation.

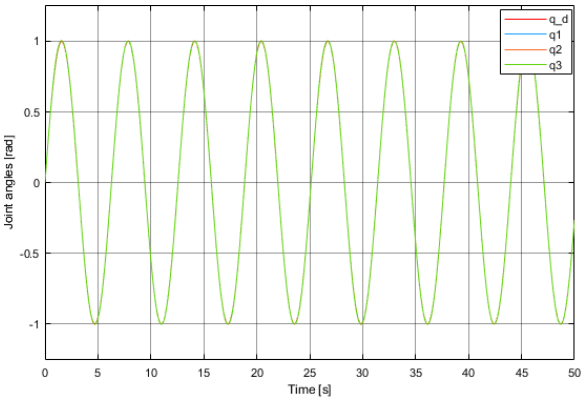



Figure 8.7: Measured joint angles versus desired joint angles using PD-controller with gravity compensation

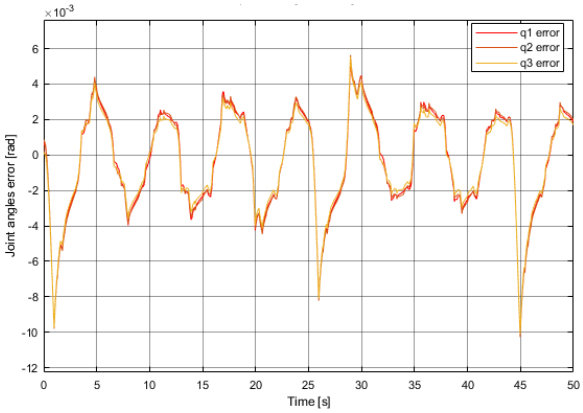



Figure 8.8: Error between desired and measured joint angles using PD-controller with gravity compensation

Maximal error between desired and measured joint angles is found from Figure 8.8 and the values for each joint variable are:

- $q_1$ error= $-1.023 \cdot 10^{-2}[rad]$ at time $t = 44.968[s]$
- $q_2$ error=$-1.027 \cdot 10^{-2}[rad]$ at time $t = 44.968[s]$
- $q_3$ error=$-1.010 \cdot 10^{-2}[rad]$ at time $t = 44.967[s]$

### 8.1.5 PI-controller

Results from simulation of the PI-controller design include measured joint angles versus desired joint angles and the error between desired and measured joint angles. Controller parameters for the PI-controller design are also found from testing several values. The best results were obtained when $K_p$ and $K_i$ where increased to their limits, which eventually resulted in very high gains.

Figure 8.9 shows the desired and measured joint angles and Figure 8.10 shows the error as the difference between desired and measured joint angles.

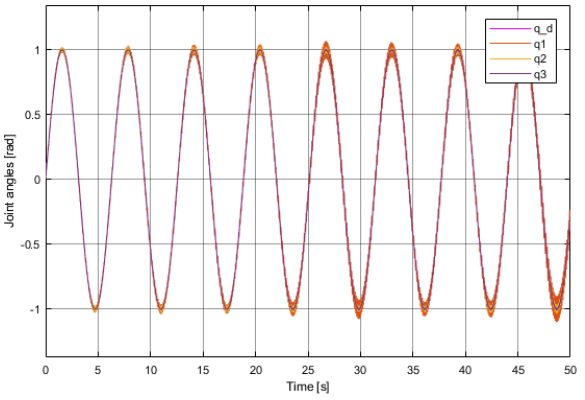



Figure 8.9: Measured joint angles versus desired joint angles using PI-controller

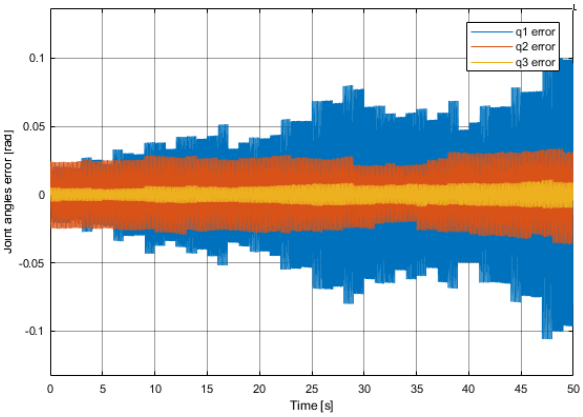



Figure 8.10: Error between desired and measured joint angles using PI-controller

Maximal error between desired and measured joint angles is found from Figure 8.10 and the values for each joint variable are:

- $q_1$ error= $1.097 \cdot 10^{-1}[rad]$ at time $t = 47.986[s]$
- $q_2$ error= $-3.675 \cdot 10^{-2}[rad]$ at time $t = 49.630[s]$
- $q_3$ error= $1.107 \cdot 10^{-2}[rad]$ at time $t = 47.992[s]$

### 8.1.6 PI-controller with Gravity Compensation

Results from simulation of the PI-controller with gravity compensation design include measured joint angles versus desired joint angles and the error between desired and measured joint angles.

Using the same controller parameters as for the PI-controller design, Figure 8.11 shows the desired and measured joint angles and Figure 8.12 shows the error as the difference between desired and measured joint angles, when adding gravity compensation.

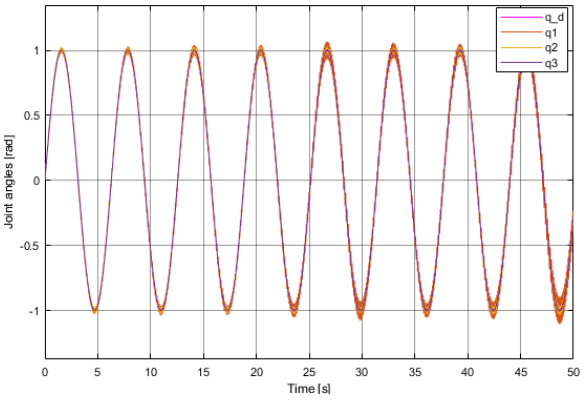



Figure 8.11: Measured joint angles versus desired joint angles using PI-controller with gravity compensation

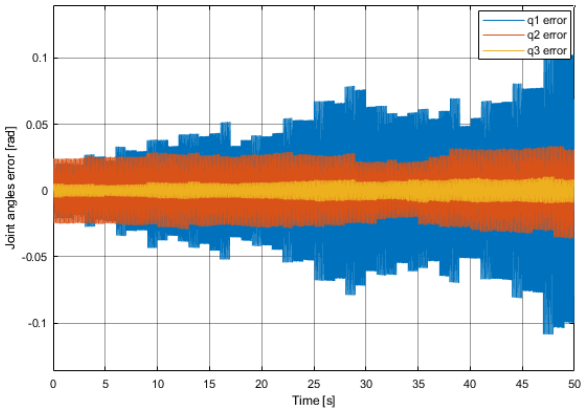



Figure 8.12: Error between desired and measured joint angles using PI-controller with gravity compensation

Maximal error between desired and measured joint angles is found from Figure 8.12 and the values for each joint variable are:

- $q_1$ error= $1.122 \cdot 10^{-1}[rad]$ at time $t = 47.986[s]$
- $q_2$ error= $-3.630 \cdot 10^{-2}[rad]$ at time $t = 49.5[s]$
- $q_3$ error= $1.061 \cdot 10^{-2}[rad]$ at time $t = 47.251[s]$

### 8.1.7 LQR

Results from simulation of the LQR design include measured joint angles versus desired joint angles and the error between desired and measured joint angles.

Before the results from control of joint 1 are presented the optimal state-feedback gain and pre-filter are calculated to be

$$K_1 = [155.7353 \ -4.0947 \ 3.7728 \ -2.6872]$$

$$V_1 = 4.9491 \cdot 10^4$$

With the use of the values above, Figure 8.13 and 8.14 show the desired and measured joint angle 1 and the error as the difference between desired and measured joint angle 1, respectively.

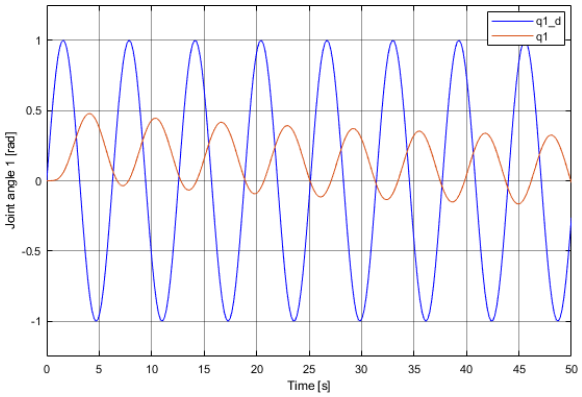



Figure 8.13: Measured joint angle 1 versus desired joint angle 1 using LQR

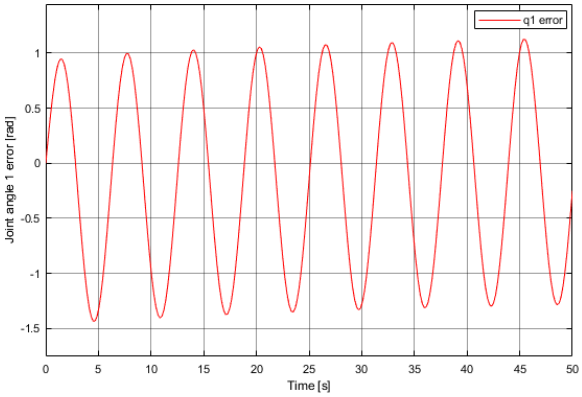



Figure 8.14: Error between desired and measured joint angle 1 using LQR

Before the results from control of joint 2 are presented the optimal state-feedback gain and pre-filter are calculated to be

$K_2$=[-59.7851 1.5899 -1.4547 1.0295]

$V_2$=-0.8221

With the use of the values above, Figure 8.15 and 8.16 show the desired and measured joint angle 2 and the error as the difference between desired and measured joint angle 2, respectively.

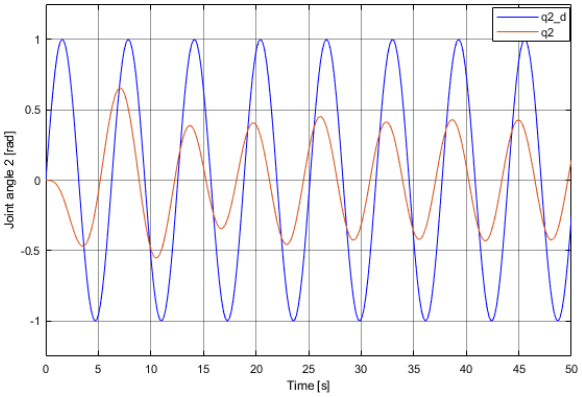



Figure 8.15: Measured joint angle 2 versus desired joint angle 2 using LQR

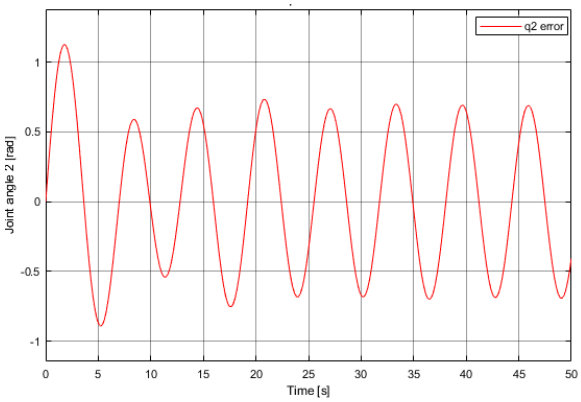



Figure 8.16: Error between desired and measured joint angle 2 using LQR

Before the results from control of joint 3 are presented the optimal state-feedback gain and pre-filter are calculated to be

$K_3$=[89.7819 -253.9393 127.1083 967.7061]

$V_3$=1.001

With the use of the values above, Figure 8.17 and 8.18 show the desired and measured joint angle 3 and the error as the difference between desired and measured joint angle 3, respectively.

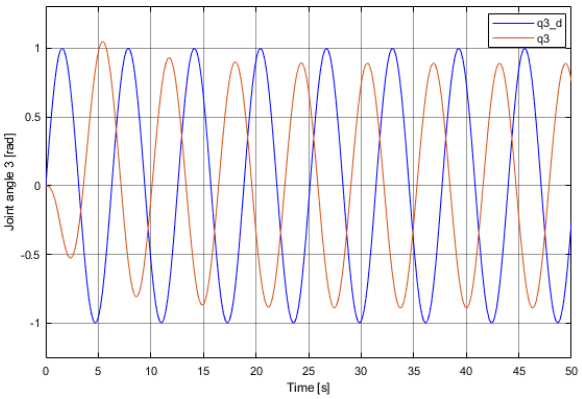



Figure 8.17: Measured joint angle 3 versus desired joint angle 3 using LQR

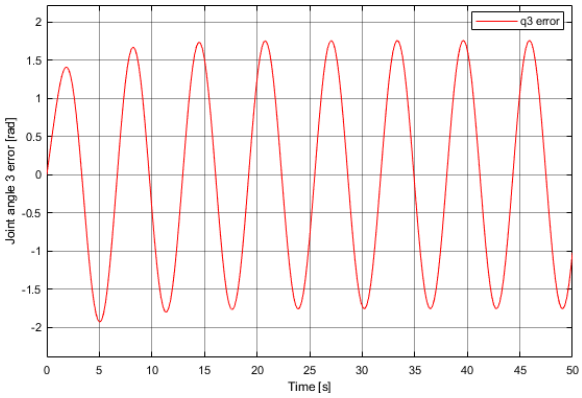



Figure 8.18: Error between desired and measured joint angle 3 using LQR

Maximal error between desired and measured joint angles is found from Figure 8.14, 8.16 and 8.18 and the values for each joint variable are:

- $q_1$ error= $-1.434[rad]$ at time $t = 4.505[s]$
- $q_2$ error=$1.127[rad]$ at time $t = 1.751[s]$
- $q_3$ error=$1.760[rad]$ at time $t = 45.920[s]$

### 8.1.8 Discussion of Control of Joints

A PID-controller design manages to track the desired joint angles well, and the error between desired and measured joint angles is small for all joints. A PID-controller with gravity compensation shows almost identical results. The error is a bit larger for all joints, but the difference is so minimal that adding the gravity term to the PID-controller designs has almost no impact on the results. Removing the derivative term of the PID-controller results in a larger error using a PD-controller design. That being said, this controller design also manages to track the desired joint angles for all joints well, as the error is minimally larger. Adding the gravity term to this controller design has an impact to the system response, as the error become larger. Removing the integral term from the PID-controller has greater impact of the results, as the error using a PI-controller design cause a significantly larger error. Using this controller design also causes an oscillating system response. The measured joint angles always oscillate around the desired joint angle. Adding a gravity term to this controller has almost no impact on the results, as the error is minimally larger but almost identical. The LQR design seems to have problems of tracking the desired joint angles for all three joints. One reason might be because of the estimated state-space models, and that using the estimated model as the plant might cause inaccuracies.

## 8.2 Control of Crane End-effector

Finally, results from simulation of the control designs concerning control of crane end-effector will be presented. This consist of results from controller designs as follows

- PID-controller with gravity compensation
- PD-controller with gravity compensation
- PI-controller with gravity compensation
- LQR

- Jacobian inversion method
- Jacobian transpose method

The measured end-effector position in z-direction will be compared to the desired end-effector position in z-direction to examine the ability each controller design has to track the desired position. When the control task concerning control of crane end-effector is to get the end-effector to follow a desired end-effector position, there will occur a certain error as the difference between the desired and measured end-effector position. This error will be presented in form of a graph with respect to time and with a maximal value. To fulfill the control task, controller parameters need to be found or calculated. The resulting parameters are also presented in this chapter.

### 8.2.1 PID-controller

Before results from simulation of the PID-controller design, calculated values of the controller parameters $K_p$, $K_i$ and $K_d$ will be presented. Then measured end-effector position versus desired end-effector position and the error between the desired and measured end-effector will be presented with Ziegler-Nichols controller parameters and with adjusted gains.

Controller parameters are found from Ziegler Nichols closed-loop tuning method. From this method the ultimate gain became $K_u = 1600$, which resulted in the following controller parameter for a PID:

$K_p = 900$

$K_i = 28.125$

$K_d = 7200$

Results from the PID-controller design with the calculated controller parameters are shown below, where Figure 8.19 shows the measured and desired position in z-direction and Figure 8.20 shows the error as the difference between desired and measured position.

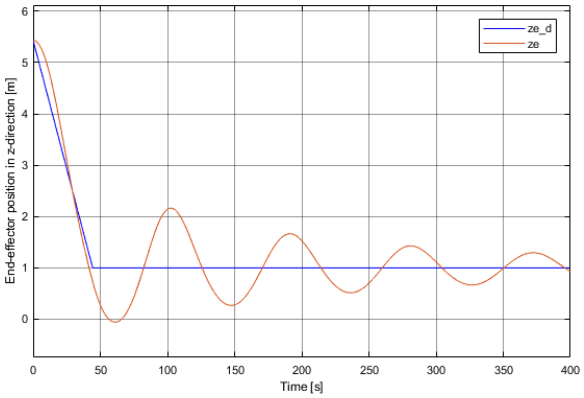



Figure 8.19: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PID-control with gravity compensation and with the use of Ziegler-Nichols parameters

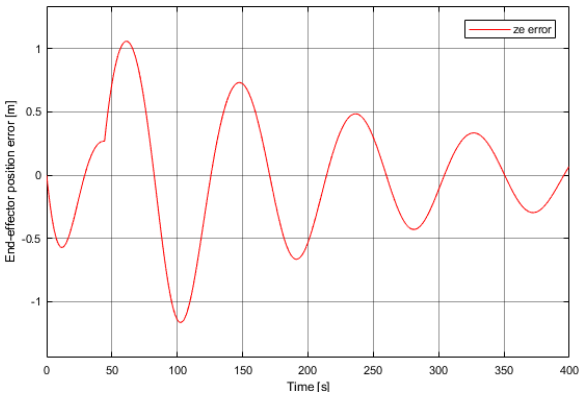
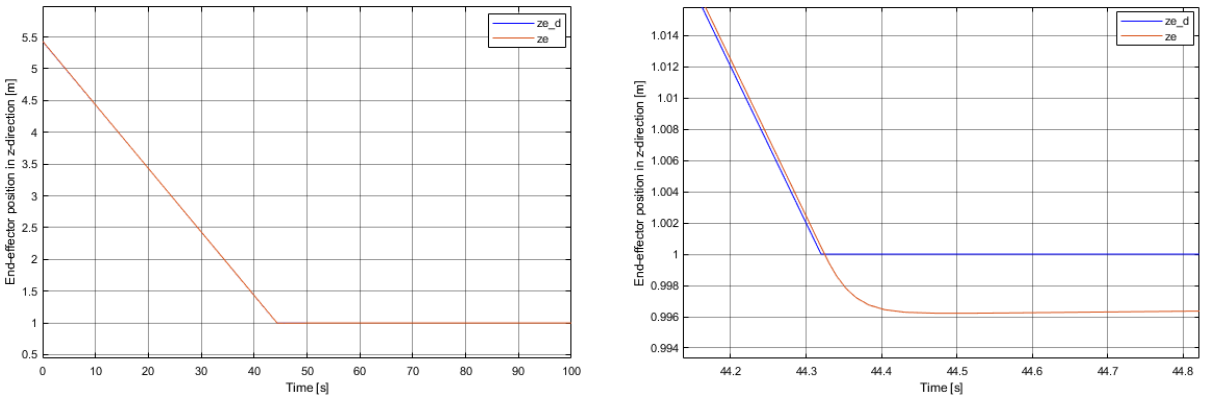



Figure 8.20: Error between desired and measured end-effector position in z-direction using PID-control with gravity compensation and with the use of Ziegler-Nichols parameters

As seen in Figure 8.20, the maximal error between desired and measured end-effector position is over $\pm 1m$. To obtain a smaller error the controller parameters were adjusted. The error

decreased when $K_p$ and $K_d$ were further increased, and new results with adjusted controller parameters are shown in Figure 8.21 and 8.22.




(a) (b)

Figure 8.21: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PID-control with gravity compensation and with the use of increased gains

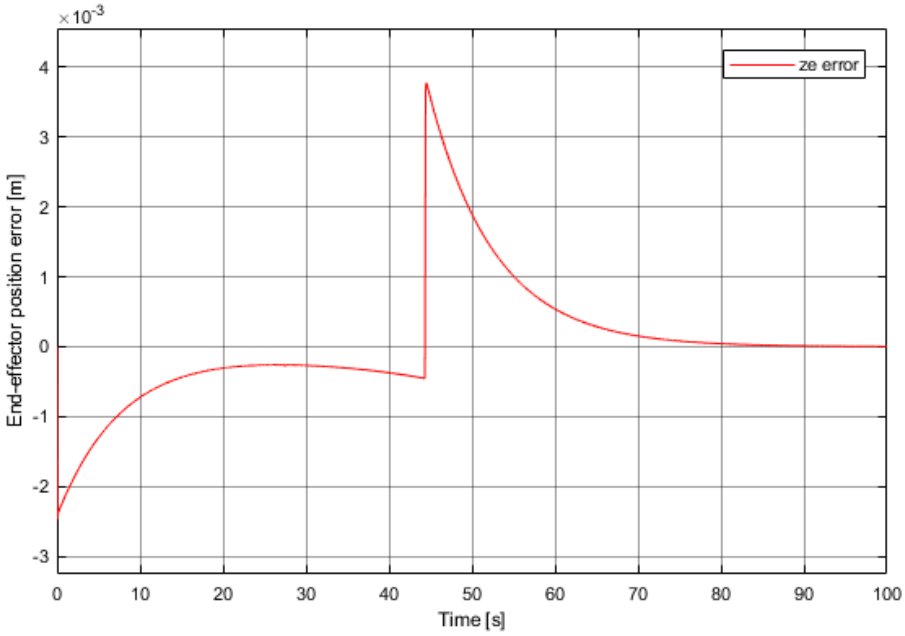



Figure 8.22: Error between desired and measured end-effector position in z-direction using PID-control with gravity compensation and with the use of increased gains

Maximal error between desired and measured end-effector position is found from Figure 8.22 and the value is:

- $z_e$ error= $3.777[mm]$ at time $t = 44.475[s]$

### 8.2.2 PD-controller

Before results from simulation of the PD-controller design, calculated values of the controller parameters $K_p$ and $K_d$ will be presented. Then measured end-effector position versus desired end-effector position and the error between desired and measured end-effector will be presented with Ziegler-Nichols controller parameters and with adjusted gains.

Controller parameters are found from Ziegler Nichols closed-loop tuning and have the following values for a PD:

$K_p$ = 1200

$K_d$ = 9600

Results from the PD-controller design with the calculated controller parameters are shown below, where Figure 8.23 shows the measured and desired position in z-direction and Figure 8.24 shows the error as the difference between desired and measured position.

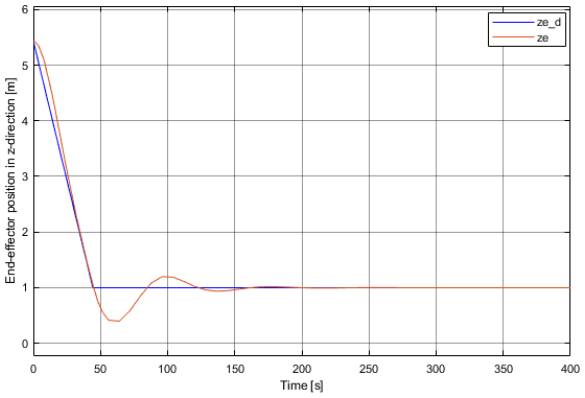



Figure 8.23: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PD-control with gravity compensation and with the use of Ziegler-Nichols parameters

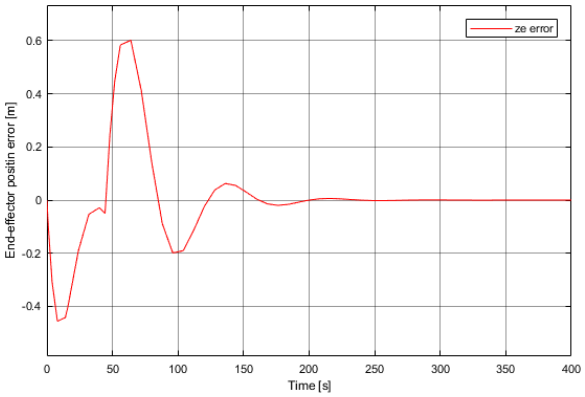
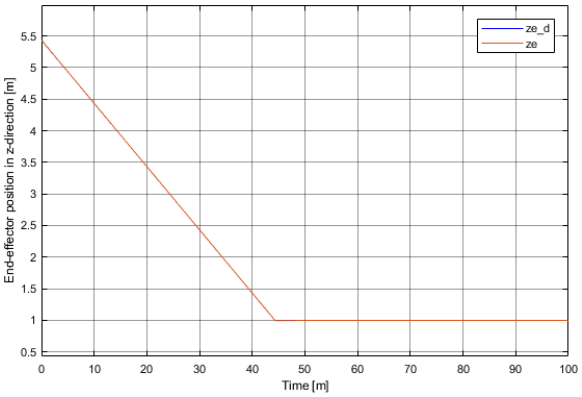
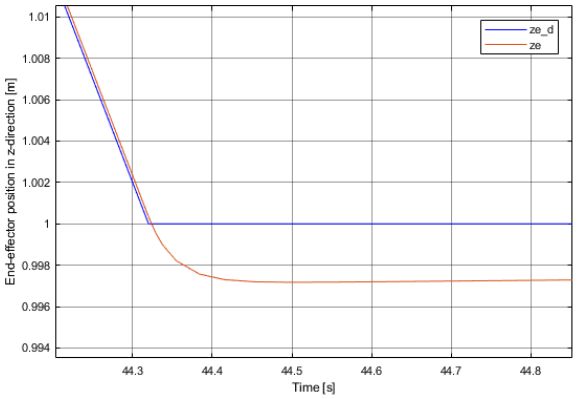



Figure 8.24: Error between desired and measured end-effector position in z-direction using PD-control with gravity compensation and with the use of Ziegler-Nichols parameters

As seen in Figure 8.24, the maximal error between desired and measured end-effector position is almost $0.6m$. To obtain a smaller error the controller parameters were adjusted. The error decreased when $K_p$ and $K_d$ were further increased, and new results with adjusted controller parameters are shown in Figure 8.25 and 8.26.




(a)




(b)

Figure 8.25: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PD-control with gravity compensation and with the use of increased gains

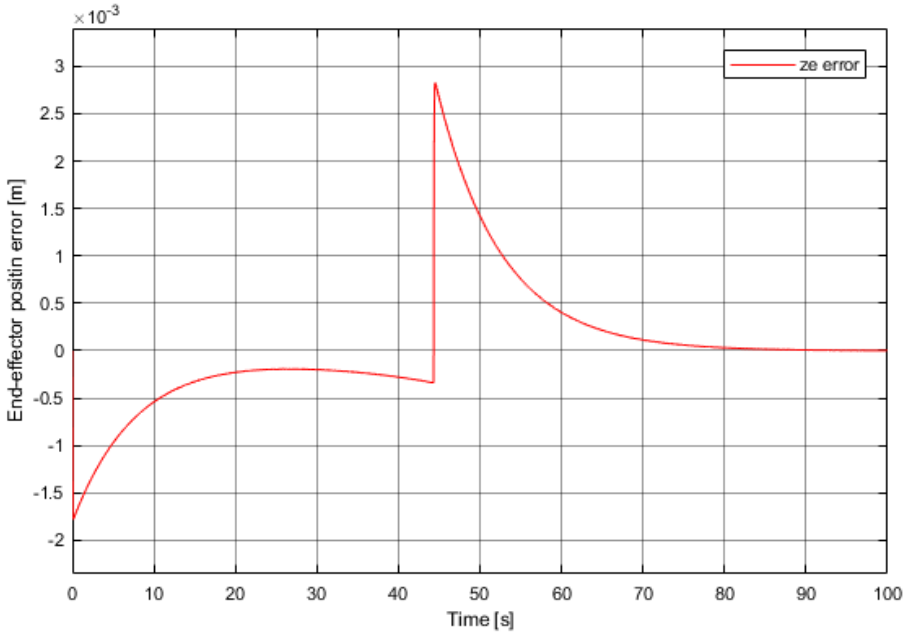



Figure 8.26: Error between desired and measured end-effector position in z-direction using PD-control with gravity compensation and with the use of increased gains

Maximal error between desired and measured end-effector position is found from Figure 8.26 and the value is:

- $z_e$ error= 2.826$[mm]$ at time $t$ = 44.498$[s]$

### 8.2.3 PI-controller

Before results from simulation of the PI-controller design, calculated values of the controller parameters $K_p$ and $K_i$ will be presented. Then measured end-effector position versus desired end-effector position and the error between desired and measured end-effector will be presented with Ziegler-Nichols controller parameters and with adjusted gains.

Controller parameters are found from Ziegler Nichols closed-loop tuning and have the following values for a PI:

$K_p$ = 675

$K_i$ = 12.66

Results from the PI-controller design with the calculated controller parameters are shown below, where Figure 8.27 shows the measured and desired position in z-direction and Figure 8.28 shows the error as the difference between desired and measured position.

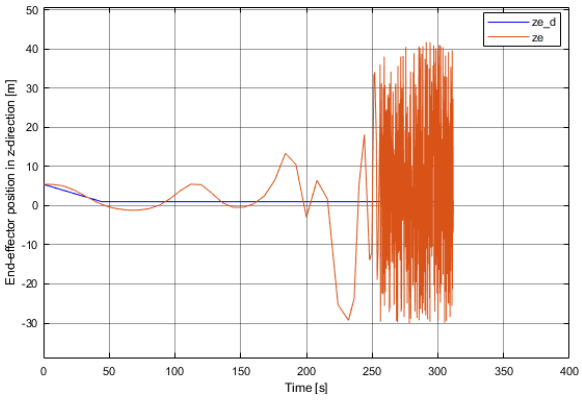



Figure 8.27: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PI-control with gravity compensation and with the use of Ziegler-Nichols parameters

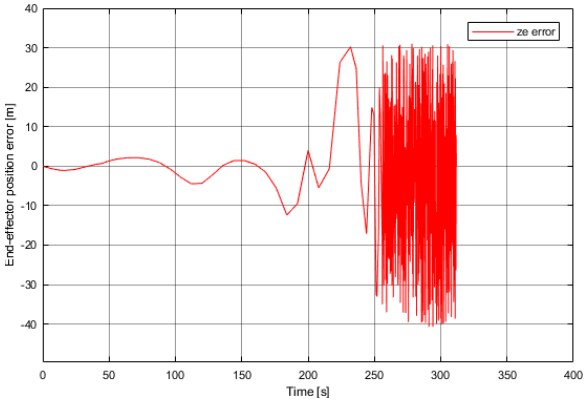
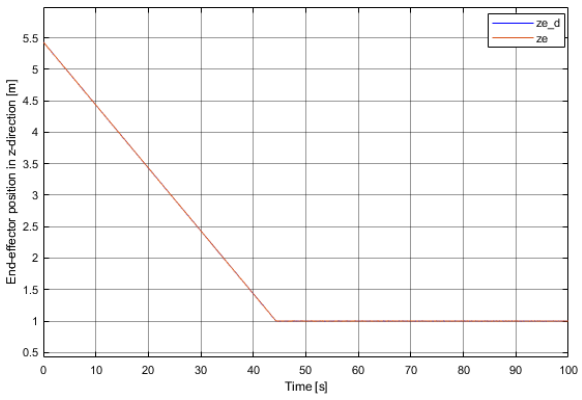
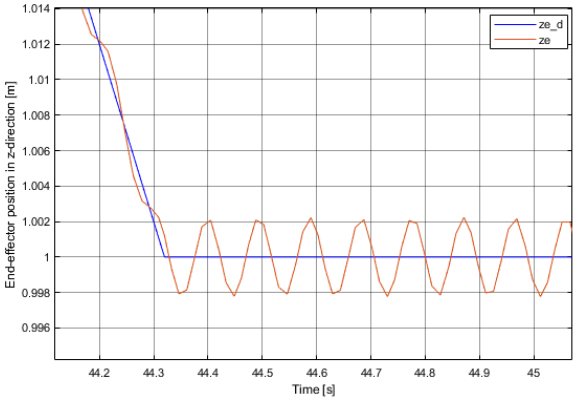



Figure 8.28: Error between desired and measured end-effector position in z-direction using PI-control with gravity compensation and with the use of Ziegler-Nichols parameters

As seen in Figure 8.28, the maximal error between desired and measured end-effector position is not measurable after a certain time using a simulation time of 400 seconds. This means that the error will increase as the time increase, and it will never be possible to track the desired position. It will be possible to obtain a smaller error with a simulation time of 100 seconds. The error decreased when $K_p$ was further increased. Results when adjusting the proportional gain are shown in Figure 8.29 and 8.30.




(a)




(b)

Figure 8.29: Measured end-effector position in z-direction versus desired end-effector position in z-direction using PI-control with gravity compensation with the use of increased gains

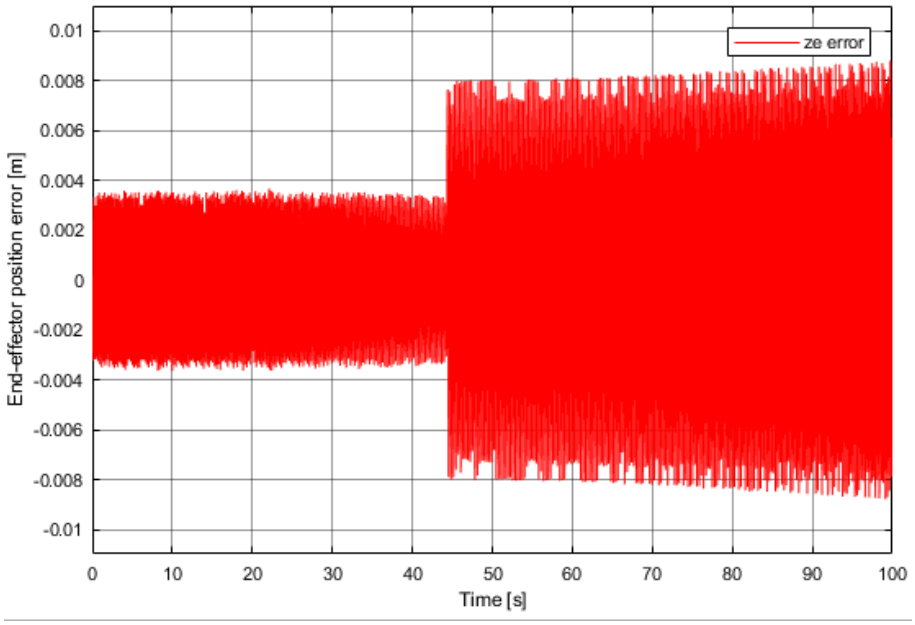



Figure 8.30: Error between desired and measured end-effector position in z-direction using PI-control with gravity compensation with the use of increased gains

Maximal error between desired and measured end-effector position is found from Figure 8.30 and the value is:

- $z_e$ error= 8.795$[mm]$ at time $t$ = 99.759$[s]$

### 8.2.4 LQR

Before results from simulation of the PID-controller design, calculated values of the optimal state feedback gain $K$ and pre-filter $V$ will be presented. Then measured end-effector position versus desired end-effector position and the error between desired and measured end-effector position will be presented.

The optimal state feedback gain and pre-filter for the LQR design are:

$K$=[-372.0925 186.3593 121.2863 59.7286]

$V$=-1.0025

Results from the LQR design are shown in Figure 8.31 and 8.32. Figure 8.31 shows how the end-effector follows the desired end-effector position, and where the error between desired and measured position is largest. Figure 8.32 shows the error as the difference between desired and measured position.

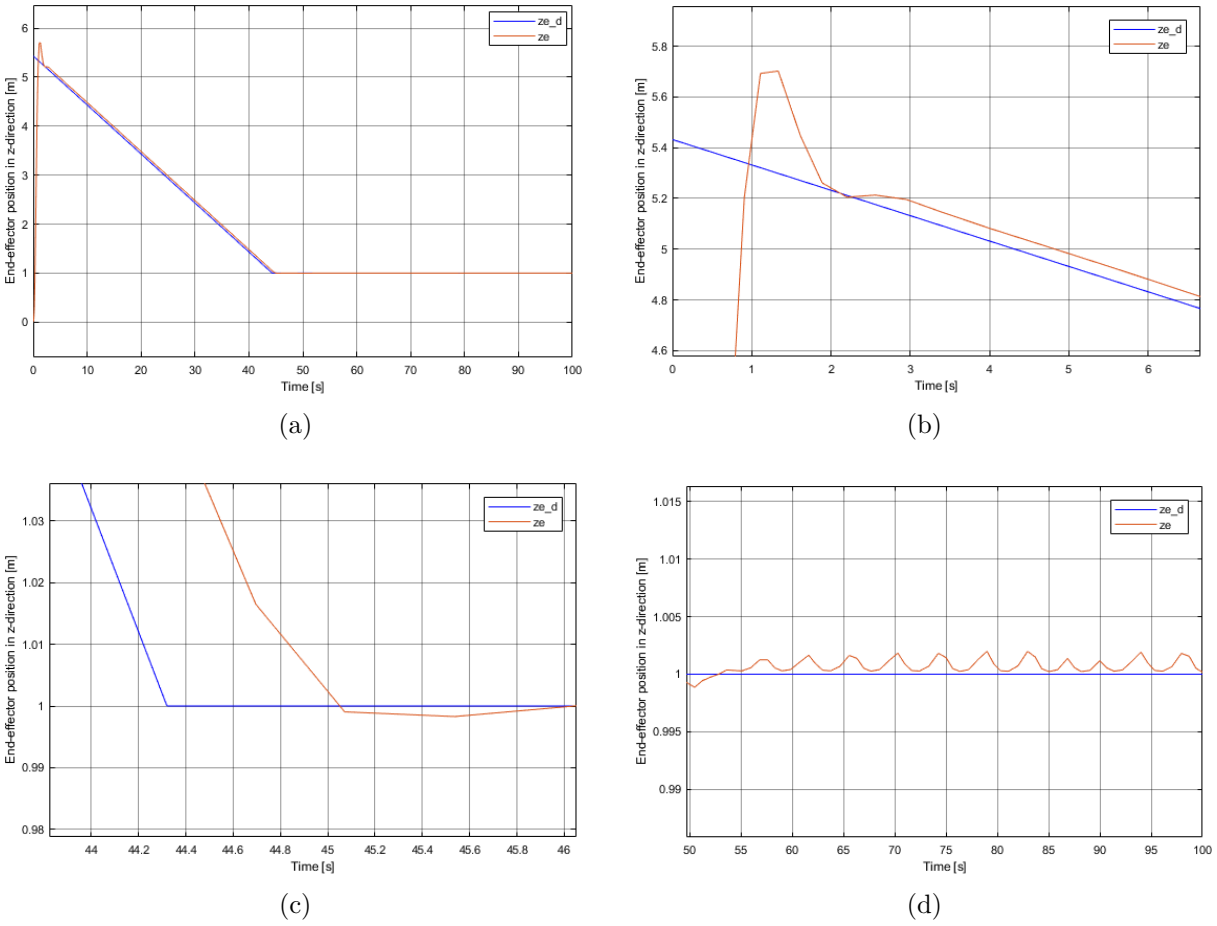



Figure 8.31: Measured end-effector position in z-direction versus desired end-effector position in z-direction using a LQR

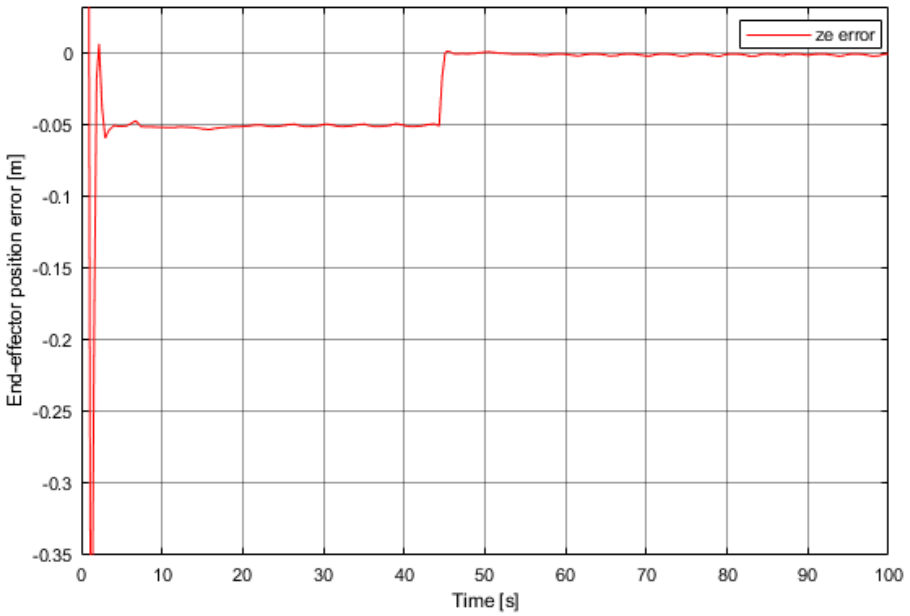



Figure 8.32: Error between desired and measured end-effector position in z-direction using a LQR

Maximal error between desired and measured end-effector position is found from Figure 8.32

and the value is:

- $z_e$ error= $5.432[m]$ at time $t = 0[s]$

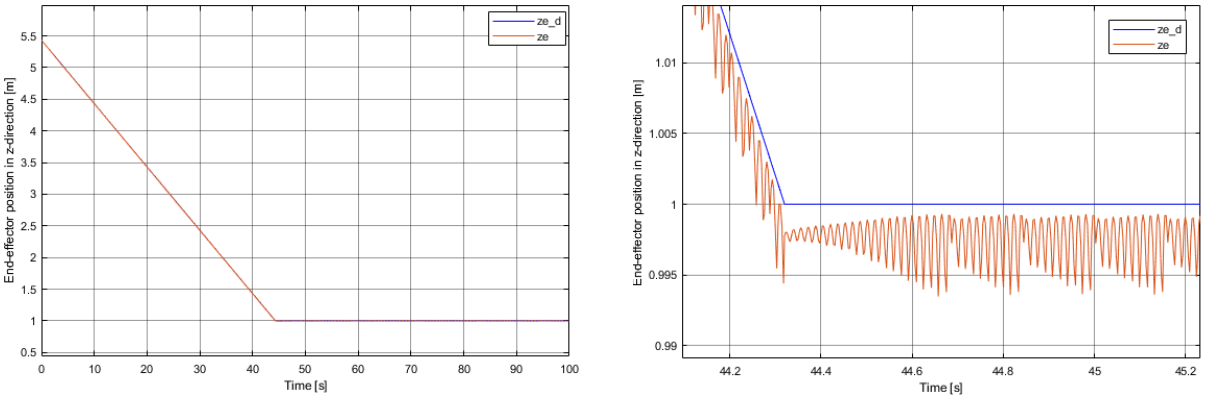
### 8.2.5 Jacobian Inversion Method

As mentioned, using the inverse of the Jacobian leads to an infinite number of solutions and singularities usually occur. Simulink was not able to run the simulation because of singularities, which means that using this method did not lead to any simulation results in this case.

### 8.2.6 Jacobian Transpose Method

Results from simulation of the Jacobian transpose controller design include measured end-effector position versus desired end-effector position and the error between desired and measured end-effector position.

Figure 8.33 shows how the measured end-effector position follows the desired end-effector position in z-direction, and Figure 8.34 shows the error as the difference between desired and measured end-effector position in z-direction.




(a) (b)

Figure 8.33: Measured end-effector position in z-direction versus desired end-effector position in z-direction using Jacobian transpose method

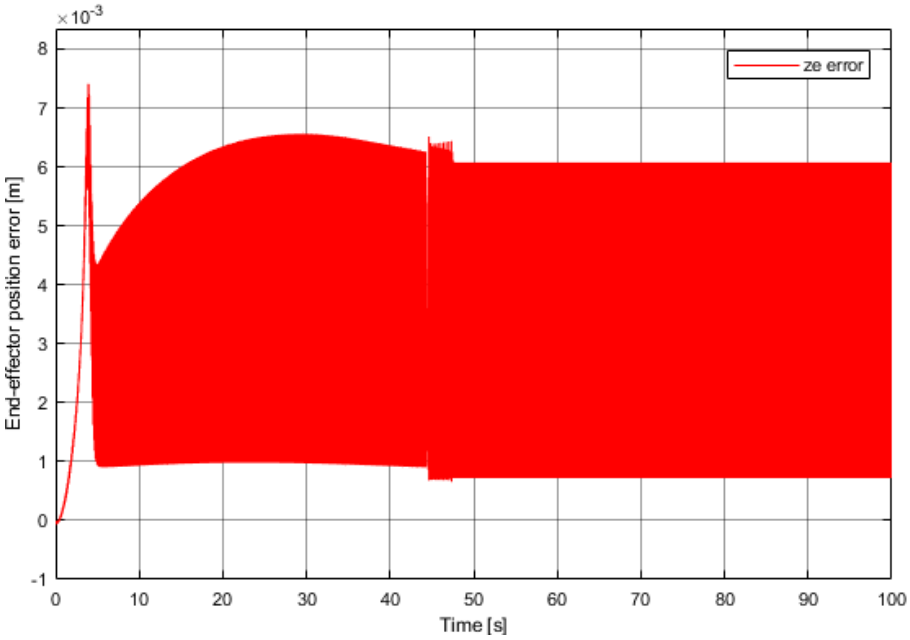



Figure 8.34: Error between desired and measured end-effector position in z-direction using Jacobian transpose method

Maximal error between desired and measured end-effector position is found from Figure 8.34 and the value is:

- $z_e$ error= $7.411[mm]$ at time $t = 3.911[s]$

### 8.2.7 Discussion of Control of Crane End-effector

A PID-controller design manages to track the desired end-effector position, and the error between desired and measured position is small. Removing the integral term tends to help reducing the error, as the results of the PID-controller design shows better results with a smaller error, which means that a PD-controller manage to track the desired end-effector position even better than the PID-controller design. Results from both PID and PD show that the error is largest when the end-effector are supposed to settle at 1m. This makes sense since the controllers must "work harder" to force the change in end-effector position. Removing the derivative term from the PID-controller has a greater impact on the results, and the PI-controller design is not able to track the desired end-effector position without oscillating, and the error is significantly larger for this control design. It can also be seen from Figure 8.30 that the error increases as the time increases, which means that it might not be possible to simulate the system after 100 seconds. The largest error when using a LQR design is in the beginning. The measured position starts at 0m, and the controller manage to force the end-effector up to the desired start position but not without overshooting. At this position the error is significantly large as well. When the end-effector reaches the desired end position of 1m, the error is larger than the errors using a PID- or a PD-controller, thus the LQR design seems to have problems of tracking the desired end-effector position compared to a PID- and PD-controller. One reason might be because of the estimated state-space model, and that the use of this model as the plant might cause inaccuracies. The Jacobian inversion method gave no results since singularities occurred, but by using the transpose Jacobian, instead of the inverse, it was possible to obtain results from the Jacobian transpose design. The maximum error is small, but the measured end-effector position always oscillates under the desired end-effector position during the whole simulation.

# Chapter 9

# Conclusions and Future Work

## 9.1 Conclusions

The purposes of this thesis has been to develop a crane model and then develop several controller designs with the task of tracking a desired crane motion.

A mathematically model of the MacGregor AHC 250t crane was developed using robot modeling theory. The crane model consisting of kinematic and dynamic equations was developed in both Matlab and Matlab/Simulink with the intention of comparing the two models. The two models showed identical results, but since these models were based on equations, a further verification could have been done by comparing the model with for instance a SimMechanics or SimulationX model.

A dynamic model of the crane was developed in Simulink with the purpose to function as a plant for several control systems. Some parts from the mathematically model consisting of the crane kinematics were also used with the purpose of controlling the crane.

Several control designs were developed with the task of tracking desired joint angles. PID-, PD- and PI- controller designs, with and without gravity compensation, were developed to control the joint angles by calculating an error value as the difference between desired and measure joint angle, and apply changes based on the controller parameters. A LQR design was developed to control the joint angles separately with the use of estimated state-space models for each joints. Several control designs were also developed with the task of tracking a desired end-effector position in vertical direction. PID-, PD- and PI- controller designs were developed to control the end-effector by calculating an error value as the difference between desired and measure end-effector position, and apply changes based on the controller parameters. A LQR design was developed to control the end-effector with the use of an estimated-state space model. Two inverse kinematics control designs were developed to minimize the error as the difference between desired and current end-effector position, using the Jacobian matrix.

A PID-controller design and a PD-controller design turned out to track the desired position with the smallest error for both control of joints and control of end-effector. A PI-controller design caused an oscillating system for both control of joints and control of end-effector. A LQR design seemed to have problems of tracking the desired joint angles and end-effector position. The Jacobian transpose design provided no results since singularities occurred and the Jacobian transpose method caused an oscillating system response.

Based on the simulations done in this thesis it can be concluded that a PID-controller design showed the best performance concerning control of crane joints, while a PD-controller design showed the best performance concerning control of end-effector.

## 9.2 Further Work

Further work on this thesis can include improvement of the LQR controller design by controlling the poles using pole-placement method. It can also be interesting to see how adding an integral term and maybe an estimator state can affect the results. Another area of improvement can be to try different desired positions and see if the controller designs provide sufficient results using a more advanced trajectory generation. In this thesis, the control designs are studied separately, but it can also be interesting to examine the use of a combination of various control design.

An offshore crane is affected by the coupled dynamics between the crane, vessel, cable and payload. Since this thesis only concerns the crane, further work can be to examine and simulate the couple dynamics of the crane, vessel, cable and payload. The implementation of various control algorithms during heave compensation is also interesting and relevant to further examine. Since control of the crane end-effector is completed the next step can be to connect a cable and payload model to the crane and further control the payload position, with the task to move the payload avoiding payload swing during the operation.

# Bibliography


[1] Thuong, K.T. and Langen, I. *Modelling and Simulation of Offshore Crane Operations on a Floating Production Vessel*. Japan, 2002.

[2] Syvertsen, P.G.Modeling and Control of Crane on Offshore Vessel. NTNU, Trondheim, Norway, 2011.

[3] Chu, Y. *Virtual Prototyping for Marine Crane Design and Operations*. NTNU, Trondheim, Norway, 2017.

[4] Chu, Y., Æsøy, V., Ehlers S. and Zhang H. *Integrated multi-domain system modelling and simulation for offshore crane operations*. University of Duisburg-Essen, Germany, 2015

[5] Chu, Y., Æsøy,V. *A multi-body dynamic model based on bond graph for maritime hydraulic crane operations*. St. John's, Newfoundland, Canada, 2015.

[6] Bertsch,C., Ahle,E., Schulmeister, U. *The Functional Mockup Interface seen from an industrial perspective*. Lund, Sweden, 2014.

[7] Rokseth, B., Skjong, S. and Pedersen, E. *Modeling of Generic Offshore Vessel in Crane Operations With Focus on Strong Rigid Body Connections*. Norway, 2017.

[8] openmodelica.org. `https://openmodelica.org/images/docs/Modelica-and-OpenModelica-overview-Peter-Fritzson-120328.pdf`. Acceded: 2018-01-08.

[9] Sun, N., Fang, Y., Chen, H., Fu, Y. and Lu, B. *Nonlinear Stabilizing Control for Ship-Mounted Cranes With Ship Roll and Heave Movements*. China, 2017.

[10] Atique, M.U and Ahad, A.R. *Inverse Kinematics Solution for a 3DOF Robotic Structure using Denavit-Hartenberg Convention*. Bangladesh, 2014.

[11] Landsverk, R. *Modelling and simulation of a knuckle boom crane and marine craft*. University of Agder, Norway, 2017. "MODELLING AND SIMULATION OF A KNUCKLE BOOM CRANE AND MARINE CRAFT"

[12] Spong, M.W, Hutchinson, S. and Vidyasagar, M. *Robot Dynamics and Control 2th Edition*. 2004.

[13] Temel, S., YAĞLI, S. and GÖREN, S. *P-PD-,PI- and PID-CONTROLLERS*. 2013.

[14] PID control. Desborough Honeywell, 2000.

[15] Control tutorials for Matlab and Simulink. *Introduction: PID Controller Design*. `http://ctms.engin.umich.edu/CTMS/index.php?example=Introduction&section=ControlPID`. Acceded: 2018-04-02.

[16] Nise, N.S. *Nises control systems 7th edition*. John Wiley & Sons, Singapore, 2015. Page 649-704.

[17] Wu, Z. and Soong, T.T. *Modified BANG-BANG Control Law for Structural Control Implementation*. New York, USA, 1996.

[18] Mohammadbagheri, A., Zaeri, N. and Yaghoobi, M. *Comparison Performance Between PID and LQR Controllers for 4-leg Voltage-Source Inverters*. Iran, 2014.

[19] Fossen, T. I. *Handbook of Marine Craft Hydrodynamics and Motion Control First Edition*. John Wiley & Sons, 2011. Chapter 12 and 13.

[20] Control tutorials for Matlab and Simulink. *Inverted Pendulum: State-Space Methods for Controller Design*. Acceeded: 2018-04-09.

[21] Barinka, L. and Berka, I.R. *Inverse Kinematics - Basic Methods*. Czech Technical Universiy Prague, Czech Republic.

# Appendix A

# Inverse Crane Kinematics and Crane Dynamics

This Appendix contains a Matlab script and two Simulink models of the inverse crane kinematics and dynamics. Both models use the same circular trajectory where the purpose is to find the motion of the joints and cylinders, and further calculate the joint torques based on the trajectory generation. The Matlab script contains of both crane kinematics and crane dynamics and shows the whole operation from the trajectory to the joint torques. The mathematical crane model is divided into two Simulink models simply to avoid the structure of the model to become to "disorderly". The first Simulink model consists of the crane kinematics and shows the operation from the trajectory to the cylinder motion. In the second Simulink model crane dynamics is included, and it shows the operation from the trajectory to the joint torques.

```
clear;
close all;
clc;

%__________initial_data
ti = 0; %Initial time
t=ti; %Time
tf = 30; %Final time
dt = 0.1;
ReportInterval=1;
Counter=ReportInterval;
ReportCounter=0;


%______Given data______

d1=5.432; %Link offset; variable joint parameter for joint 1
   5.432
l2=25.202; %Length of the second link 25.202
l3=11.182; %Length og the third link 11.182


r = 0.5;
v=0.1;
center = [15 0 0];

```

```
%____Values of C-vectors that were calculated in trajectory_x,_y
   and _z
%(polynominal trajectory)


while t<tf




%________Trajectory generation (end-effector)
%____Position of end-effector_____

omega=v/r;
theta = omega*t;
position=center+r*[cos(theta) sin(theta) zeros(size(theta))];
velocity=[cos(theta) sin(theta) zeros(size(theta))]+v*[-sin(theta
   ) cos(theta) zeros(size(theta))];
% velocity=v*[-sin(theta) cos(theta) zeros(size(theta))];
acceleration=v/r*[-sin(theta) cos(theta) zeros(size(theta))]+v/t
   *[-sin(theta) cos(theta) zeros(size(theta))]+v^2/r*[-cos(theta)
    sin(theta) zeros(size(theta))];
% acceleration=v/t*[-cos(theta) -sin(theta) zeros(size(theta))];
xe=(position(1,1));
ye=(position(1,2));
ze=d1;


%_____Velocity of end-effector____
xeDot=(velocity(1,1));
yeDot=(velocity(1,2));
zeDot=0;


%____Acceleration of end-effector_____
xeDotDot=(acceleration(1,1));
yeDotDot=(acceleration(1,2));
zeDotDot=0;

%%%%%% Inverse kinematics to find the joint angles %%%%%%%%


% file:///D:/Lisas%20minnepenn/Master%20thesis/Support%20material
   /From%20internet/Modeling_of_a_3DOF_robot_DH.pdf
%%%%%% Inverse kinematics to find the joint angles %%%%%%%%



q1=atan2(ye,xe);


```

```
c3=(xe^2+ye^2+(ze-d1)^2-l2^2-l3^2)/(2*l2*l3);
s3=sqrt(1-c3^2);

q3=atan2(s3,c3);



q2=atan2(ze-d1,sqrt(xe^2+ye^2))-atan2(l3*sin(q3),l2+l3*cos(q3));




%Joint angle vector
q=[q1;...
    q2;...
    q3];

%%%%%% Velocity of joints %%%%%%%

ve=[xeDot;...
    yeDot;...
    zeDot];

J=[-sin(q1)*(l2*cos(q2)+l3*cos(q2+q3)) -cos(q1)*(l2*sin(q2)+l3*
   sin(q2+q3)) -l3*cos(q1)*sin(q2+q3);...
    cos(q1)*(l2*cos(q2)+l3*cos(q2+q3)) -sin(q1)*(l2*sin(q2)+l3*
       sin(q2+q3)) -l3*sin(q1)*sin(q2+q3);...
    0 l2*cos(q2)+l3*cos(q2+q3) l3*cos(q2+q3)];

qDot=inv(J)*ve;
q1Dot=qDot(1,1);
q2Dot=qDot(2,1);
q3Dot=qDot(3,1);

%%%%%%% Acceleration of joints %%%%%%%%

veDot=[xeDotDot;...
    yeDotDot;...
    xeDotDot];

JDot_q1=[-sin(q1)*(l2*cos(q2)+l3*cos(q2+q3)) sin(q1)*(l2*sin(q2)+
   l3*sin(q2+q3)) l3*sin(q1)*sin(q2+q3);...
    -sin(q1)*(l2*cos(q2)+l3*cos(q2+q3)) -cos(q1)*(l2*sin(q2)+l3*
       sin(q2+q3)) -l3*cos(q1)*sin(q2+q3);...
    0 0 0];

JDot_q2=[-sin(q1)*(-l2*sin(q2)-l3*sin(q2+q3)) -cos(q1)*(l2*cos(q2
   )+l3*cos(q2+q3)) -l3*cos(q1)*cos(q2+q3);...
    cos(q1)*(-l2*cos(q2)-l3*sin(q2+q3)) -sin(q1)*(l2*cos(q2)+l3*
       cos(q2+q3)) -l3*sin(q1)*cos(q2+q3);...
    0 -l2*sin(q2)-l3*sin(q2+q3) -l3*sin(q2+q3)];
```

```
117
118 JDot_q3=[l3*sin(q1)*sin(q2+q3) -l3*cos(q1)*cos(q2+q3) -l3*cos(q1)
       *cos(q2+q3);...
119     -l3*cos(q1)*sin(q2+q3) -l3*sin(q1)*cos(q2+q3) -l3*sin(q1)*cos
           (q2+q3);...
120     0 -l3*sin(q2+q3) -l3*sin(q2+q3)];
121
122
123 JDot=JDot_q1+JDot_q2+JDot_q3;
124
125 qDotDot=inv(J)*(veDot-JDot*qDot);
126
127 q1DotDot=qDotDot(1,1);
128 q2DotDot=qDotDot(2,1);
129 q3DotDot=qDotDot(3,1);
130
131
132 %%%%%% Lenght of cylinders %%%%%%%%
133
134 %____Cylinder 1______
135
136 c1_1=2.435;
137 a1=-4.390;
138 b1=10.2765;
139 c1_2=-2.9;
140
141 a1_marked=sqrt(c1_1^2+a1^2);
142 b1_marked=sqrt(b1^2+c1_2^2);
143
144 phi_1=atan2(c1_1,a1);
145 phi_2=atan2(c1_2,b1);
146 theta_2=q2-phi_1-phi_2+pi/2;
147
148 L1=sqrt(a1_marked^2+b1_marked^2-2*a1_marked*b1_marked*cos(theta_2
       ));
149
150 %______Cylinder 2_____
151
152 c2_1=-3.090;
153 a2=-8.275;
154 c2_2=-1.8639;
155 b2=-2.38134;
156
157 a2_marked=sqrt(a2^2+c2_1^2);
158 b2_marked=sqrt(b2^2+c2_2^2);
159
160 phi_3=atan2(c2_1,a2);
161 phi_4=atan2(c2_2,b2);
162 theta_3=q3+phi_4-phi_3-pi;
163 L2=sqrt(a2_marked^2+b2_marked^2-2*a2_marked*b2_marked*cos(theta_3
       ));
```

```
164
165 %%%%%% Velocity of cylinder 1 and 2 %%%%%%%%%
166
167
168 %_____Cylinder 1______
169
170 L1Dot=a1_marked*b1_marked*sin(theta_2)*q2Dot/(sqrt(a1_marked^2+
       b1_marked^2-2*a1_marked*b1_marked*cos(theta_2)));
171
172 %____Cylinder 2_____
173
174 L2Dot=a2_marked*b2_marked*sin(theta_3)*q3Dot/(sqrt(a2_marked^2+
       b2_marked^2-2*a2_marked*b2_marked*cos(theta_3)));
175
176 %%%%%%%% Langrangian dynamics to find the joint torques%%%%%%%
177
178 %___Simplifications of sin and cos
179
180 c1=cos(q1);
181 s1=sin(q1);
182 c2=cos(q2);
183 s2=sin(q2);
184 c3=cos(q3);
185 s3=sin(q3);
186 c23=cos(q2+q3);
187 s23=sin(q2+q3);
188
189 %____Given values for Kinetic and Potential energy
190
191 g=-9.81;
192
193 m1=44404;
194 I1x=2669369.3;
195 I1y=5934622.1;
196 I1z=3662369.9;
197
198 m2=60300;
199 I2x=49810.1;
200 I2y=2237575.1;
201 I2z=2216809.5;
202
203 m3=36000;
204 I3x=90455.9;
205 I3y=348040.1;
206 I3z=266224.2;
207
208 %____Equations for the inertia matrix____
209
210 M11=I1y+I2x*s2^2+I2y*c2^2+I3z*s23^2+I3y*c23^2+m2*(1/2*l2)^2*c2^2+
       m3*(1/2*c23+l2*c2)^2;
211 M12=0;
```

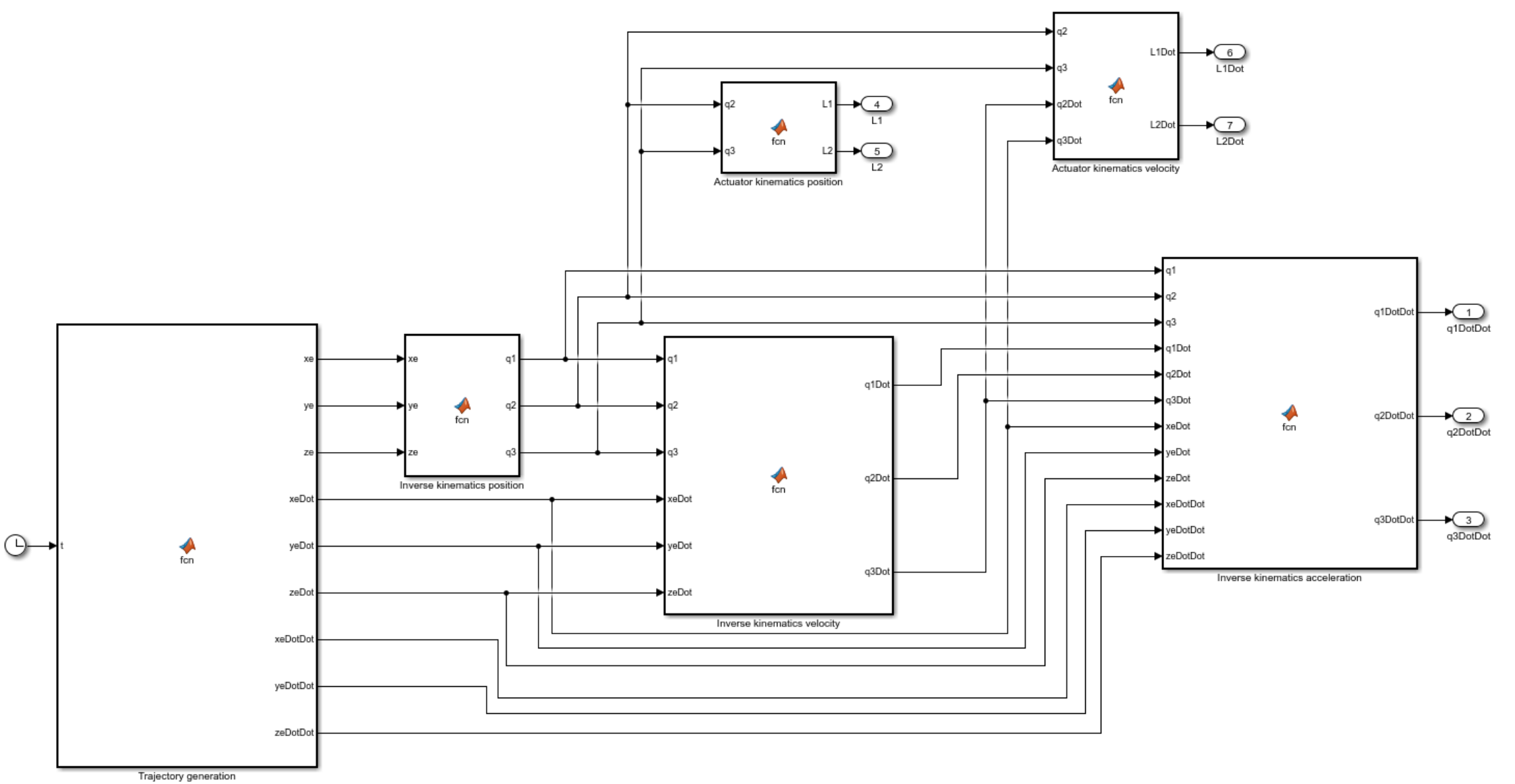
```
M21=0;
M13=0;
M31=0;
M22=I2z+I3z+m2*(1/2*l2)^2+m3*((1/2*l3)^2+l2^2+l2*l3*c3);
M23=I3z+m3*((1/2*l3)^2+1/2*l2*l3*c3);
M32=I3z+m3*((1/2*l3)^2+1/2*l2*l3*c3);
M33=I3z+m3*(1/2*l3)^2;

%_______Inertia matrix_____
M=[M11 M12 M13;...
    M21 M22 M23;...
    M31 M32 M33];


%____Equations for Coriolis and centripetal matrix______

l2c=1/2*l2;
l3c=1/2*l3;

C11=(s2*c2*(I2x-I2y)+c23*s23*(I3x-I3y)-m2*l2c^2*c2*s2-m3*(l3c*c23
   +l2*c2)*(l3c*s23+l2*s2))*q2Dot+(c23*s23*(I3x-I3y)-m3*l3c*s23*(
   l3c*c23+l2*c2))*q3Dot;
C12=(s2*c2*(I2x-I2y)+c23*s23*(I3x-I3y)-m2*l2c^2*c2*s2-m3*(l3c*c23
   +l2*c2)*(l3c*s23+l2*s2))*q1Dot;
C13=(c23*s23*(I3x-I3y)-m3*l3c*s23*(l3c*c23+l2*c2))*q1Dot;
C21=-C12;
C22=-1/2*m3*l2*l3*s3*q3Dot;
C23=-1/2*m3*l2*l3*s3*q2Dot-m3*l3c*l3*s3*q3Dot;
C31=-C13;
C32=1/2*m3*l2*l3*s3*q2Dot;
C33=0;

%____Coriolis and centripetal matrix_____

C=[C11 C12 C13;...
    C21 C22 C23;...
    C31 C32 C33];


%____Gravity vector

G=[0;...
    m2*g*l2c*c2+m3*g*l3c*c23+l2*c2;...
    m3*g*l3c*c23];

%%%% Joint torques %%%%%%

tau=M*qDotDot+C*qDot+G;

tau1=tau(1,1);
tau2=tau(2,1);
```

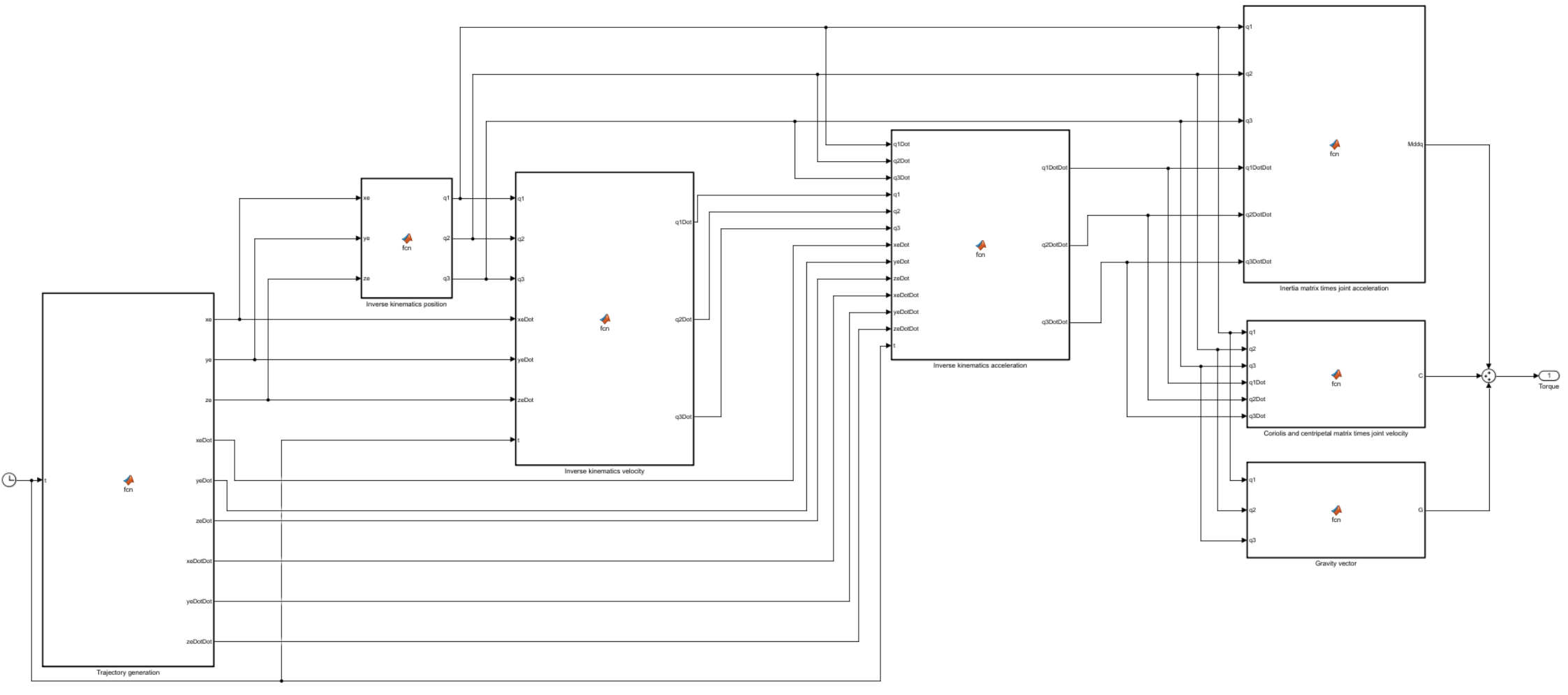
```
tau3=tau(3,1);


if Counter==ReportInterval
        Counter=0;
        ReportCounter=ReportCounter+1;
        TimePlot(ReportCounter)=t;

        %Trajectory position
        xe_Plot(ReportCounter)=xe;
        ye_Plot(ReportCounter)=ye;
        ze_Plot(ReportCounter)=ze;

        %Trajectory velocity
        xeDot_Plot(ReportCounter)=xeDot;
        yeDot_Plot(ReportCounter)=yeDot;
        zeDot_Plot(ReportCounter)=zeDot;

        %Trajectory acceleration
        xeDotDot_Plot(ReportCounter)=xeDotDot;
        yeDotDot_Plot(ReportCounter)=yeDotDot;
        zeDotDot_Plot(ReportCounter)=zeDotDot;

        %Trajectory acceleration
        q1_Plot(ReportCounter)=q1;
        q2_Plot(ReportCounter)=q2;
        q3_Plot(ReportCounter)=q3;


        %Joint velocities
        q1Dot_Plot(ReportCounter)=q1Dot;
        q2Dot_Plot(ReportCounter)=q2Dot;
        q3Dot_Plot(ReportCounter)=q3Dot;

         %Joint acceleration
        q1DotDot_Plot(ReportCounter)=q1DotDot;
        q2DotDot_Plot(ReportCounter)=q2DotDot;
        q3DotDot_Plot(ReportCounter)=q3DotDot;

        %Cylinder lengths
        L1_Plot(ReportCounter)=L1;
        L2_Plot(ReportCounter)=L2;

        %Cylinder velocties
        L1Dot_Plot(ReportCounter)=L1Dot;
        L2Dot_Plot(ReportCounter)=L2Dot;

        %Joint torque
        tau1_Plot(ReportCounter)=tau1;
        tau2_Plot(ReportCounter)=tau2;
        tau3_Plot(ReportCounter)=tau3;
```

```
311
312
313         end;
314
315 %__________Time incresment
316     i=i+1;
317     t=t+dt;
318     Counter=Counter+1;
319 end;
320
321
322 figure (1)
323 plot(TimePlot,xe_Plot)
324 hold on
325 plot(TimePlot,ye_Plot)
326 hold on
327 plot(TimePlot,ze_Plot)
328 ylabel('End-effector position [m]')
329 xlabel('Time [s]')
330 ylim([-5,16])
331 xlim([0 30])
332 legend('$x_e$','$y_e$','$z_e$')
333 set(legend('$x_e$','$y_e$','$z_e$'), 'Interpreter', 'latex');
334
335 figure (2)
336 plot(TimePlot,xeDot_Plot)
337 hold on
338 plot(TimePlot,yeDot_Plot)
339 hold on
340 plot(TimePlot,zeDot_Plot)
341 ylabel('End-effector velocity [m/s]')
342 xlabel('Time [s]')
343 ylim([-1.5,1.5])
344 xlim([0 30])
345 legend('$\dot{x}_e$','$\dot{y}_e$','$\dot{z}_e$')
346 set(legend('$\dot{x}_e$','$\dot{y}_e$','$\dot{z}_e$'), '
       Interpreter', 'latex');
347
348 figure (3)
349 plot(TimePlot,xeDotDot_Plot)
350 hold on
351 plot(TimePlot,yeDotDot_Plot)
352 hold on
353 plot(TimePlot,zeDotDot_Plot)
354 ylabel('End-effector acceleration [m/s^2]')
355 xlabel('Time [s]')
356 ylim([-0.5 0.5])
357 xlim([0 30])
358 legend('$\ddot{x}_e$','$\ddot{y}_e$','$\ddot{z}_e$')
359 set(legend('$\ddot{x}_e$','$\ddot{y}_e$','$\ddot{z}_e$'), '
       Interpreter', 'latex');
```

```
360
361 %Angles of joints
362 figure (4)
363 plot(TimePlot,q1_Plot)
364 hold on
365 plot(TimePlot,q2_Plot)
366 hold on
367 plot(TimePlot,q3_Plot)
368 ylabel('Joint angles [rad]')
369 xlabel('Time [s]')
370 ylim([-0.5 3])
371 xlim([0 30])
372 legend('$q_1$','$q_2$','$q_3$')
373 set(legend('$q_1$','$q_2$','$q_3$'), 'Interpreter', 'latex');
374
375 %Velocity of joints
376 figure (5)
377 plot(TimePlot,q1Dot_Plot)
378 hold on
379 plot(TimePlot,q2Dot_Plot)
380 hold on
381 plot(TimePlot,q3Dot_Plot)
382 ylabel('Joint velocities [rad/s]')
383 xlabel('Time [s]')
384 ylim([-0.15 0.25])
385 xlim([0 30])
386 legend('$\dot{q}_1$','$\dot{q}_2$','$\dot{q}_3$')
387 set(legend('$\dot{q}_1$','$\dot{q}_2$','$\dot{q}_3$'), '
       Interpreter', 'latex');
388
389 %Acceleration of joints
390 figure (6)
391 plot(TimePlot,q1DotDot_Plot)
392 hold on
393 plot(TimePlot,q2DotDot_Plot)
394 hold on
395 plot(TimePlot,q3DotDot_Plot)
396 ylabel('Joint accelerations [rad/s^2]')
397 xlabel('Time [s]')
398 ylim([-0.5 1])
399 xlim([0 30])
400 legend('$\ddot{q}_1$','$\ddot{q}_2$','$\ddot{q}_3$')
401 set(legend('$\ddot{q}_1$','$\ddot{q}_2$','$\ddot{q}_3$'), '
       Interpreter', 'latex');
402
403 %Lengt of cylinders
404 figure (7)
405 plot(TimePlot,L1_Plot)
406 hold on
407 plot(TimePlot,L2_Plot)
408 ylabel('Cylinder lengths [m]')
```

```
409 xlabel('Time [s]')
410 ylim([5.5 9.5])
411 xlim([0 30])
412
413 legend('$L_1$','$L_2$')
414 set(legend('$L_1$','$L_2$'), 'Interpreter', 'latex');
415
416 %Velocity of cylinders
417 figure (8)
418 plot(TimePlot,L1Dot_Plot)
419 hold on
420 plot(TimePlot,L2Dot_Plot)
421 ylabel('Cylinder velocities [m/s]')
422 xlabel('Time [s]')
423 ylim([-1 0.5])
424 xlim([0 30])
425 legend('$\dot{L}_1$','$\dot{L}_2$')
426 set(legend('$\dot{L}_1$','$\dot{L}_2$'), 'Interpreter', 'latex');
427
428 % Joint Tourqes
429 figure (9)
430 plot(TimePlot,tau1_Plot)
431 hold on
432 plot(TimePlot,tau2_Plot)
433 hold on
434 plot(TimePlot,tau3_Plot)
435 ylabel('Joint torques [Nm]')
436 xlabel('Time [s]')
437 ylim([-1.5e7 1.5e7])
438 xlim([0 30])
439 legend('$\tau_1$','$\tau_2$','$\tau_3$')
440 set(legend('$\tau_1$','$\tau_2$','$\tau_3$'), 'Interpreter', '
      latex');
```


Trajectory generation
t
fcn
xe
ye
ze
xeDot
yeDot
zeDot
xeDotDot
yeDotDot
zeDotDot
Inverse kinematics position
fcn
q1
q2
q3
Actuator kinematics position
fcn
L1
L2
4
L1
5
L2
Inverse kinematics velocity
fcn
q1Dot
q2Dot
q3Dot
Actuator kinematics velocity
fcn
L1Dot
L2Dot
6
L1Dot
7
L2Dot
Inverse kinematics acceleration
fcn
q1DotDot
q2DotDot
q3DotDot
1
q1DotDot
2
q2DotDot
3
q3DotDot


Trajectory generation
Inverse kinematics position
Inverse kinematics velocity
Inverse kinematics acceleration
Inertia matrix times joint acceleration
Coriolis and centripetal matrix times joint velocity
Gravity vector
Torque
fcn

# Appendix B

# Linear-Qaudratic Regulator (LQR) Design for Crane Joints

This Matlab script contains a LQR design concerning control of crane joints, and is run in parallel with the Simulink models from chapter 7.1.4.

```
%%%%%%%%%%%%%%%%%%%%LQR for joints%%%%%%%%%%%%%%%%%%%%%%%%


%State-space model

%Joint 1

A1=[0.02802 -0.03715 -0.0001235 -0.0003595;...
    0.08716 0.01595 -1.523 -0.1277;...
    0.0135 1.525 -0.003248 -3.92;...
    0.06509 0.00179 2.274 -5.305];

B1=[-1.875e-06;...
    -0.001049;...
    -0.006192;...
    -0.02576];

C1=[0.001536 -2.625e-06 1.954e-7 -2.66e-08];

D1=0;

Plant_q1=ss(A1,B1,C1,D1);


%Joint 2

A2=[-0.002628 -0.4077 0.005025 -0.004728;...
    0.3959 -0.03321 1.719 -0.8239;...
    -0.002582 -1.023 -3.217 5.305;...
    -0.005316 0.3527 -0.05275 -0.9313];

B2=[9.272e-06;...
    0.002335;...
    -0.05772;...
```

```
    0.006049];

C2=[40.51 -0.7789 -0.05368 0.0141];

D2=0;

Plant_q2=ss(A2,B2,C2,D2);


%Joint 3

A3=[-0.006824 -0.165 0.005018 -0.005282;...
    0.1782 -0.05213 2.492 -1.094;...
    -0.002169 -1.167 -6.102 6.494;...
    -0.005329 0.147 1.522 -0.8303];

B3=[-5.045e-05;...
    -0.01071;...
    0.1217;...
    -0.01619];

C3=[395.9 -3.079 -2.2547 0.03162];

D3=0;

Plant_q3=ss(A3,B3,C3,D3);


 % Determine whether the system is controllable
 cr_q1=ctrb(Plant_q1);
 rank_q1=rank(cr_q1);

 cr_q2=ctrb(Plant_q2);
 rank_q2=rank(cr_q2);

 cr_q3=ctrb(Plant_q3);
 rank_q3=rank(cr_q3);

%Since the rank for all systems are equalt to the order of the
   sytsem the
%systems are controllable


%Poles of the open-loop system
p_ol_q1 = eig(A1);
p_ol_q2 = eig(A2);
p_ol_q3 = eig(A3);

%Tracking error
Q1 = C1'*C1;
Q2 = C2'*C2;
```

```
85 Q3 = C3'*C3;
86
87 %Input weights
88 R1=eye(1);
89 R2=eye(1);
90 R3=eye(1);
91
92 %Compute the optimal feedback gain matrix K for the open-loop
      system
93 K1 = lqr(A1,B1,Q1,R1);
94 K2 = lqr(A1,B2,Q2,R2);
95 K3 = lqr(A3,B3,Q3,R3);
96
97 % Design pre-filter
98 V1 = (C1*(B1*K1-A1)^(-1)*B1)^(-1);
99 V2 = (C2*(B2*K2-A2)^(-1)*B2)^(-1);
100 V3 = (C3*(B3*K3-A3)^(-1)*B3)^(-1);
```

# Appendix C

# Linear-Qaudratic Regulator (LQR) Design for Crane End-effector

This Matlab script contains a LQR design concerning control of crane end-effector in z-direction, and is run in parallel with the Simulink model from chapter 7.2.5.

```
%%%%%%%%%%%%%%%%%%%%LQR for contol of end-effector in z-direction
    %%%%%%%%%%%%%%%%%%%%%%%%


%State-space model

A=[-0.0002949 -1.08 0.03363 -0.0299;...
    1.074 -0.1639 3.541 -1.236;...
    0.01761 -1.91 -3.677 4.467;...
    -0.001651 0.2363 0.5733 -0.2493];

B=[-0.0003137;...
    -0.009771;...
    0.05251;...
    0.0028];

C=[444.3 -23.12 -3.194 0.3886];

D=0;

Plant=ss(A,B,C,D);


 % Determine whether the system is controllable
 cr=ctrb(Plant);
 rank=rank(cr);

 %Since the rank is equal to the order of the system the system
    is
 %controllable



%Tracking error
```

```
33 Q = C'*C;
34
35 %Input weights
36 R=eye(1);
37
38
39 %Compute the optimal feedback gain matrix K for the open-loop
      system
40 K = lqr(A,B,Q,R);
41
42
43 % Design pre-filter
44 V = (C*(B*K-A)^(-1)*B)^(-1);
```