Master's thesis

NTNU
Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

Henrik Hammarstrøm

Autonomous Loading Operations with a Shipboard Knuckle-boom Crane

Master's thesis in Marine Technology
Supervisor: Eilif Pedersen
Co-supervisor: Tom Arne Pedersen
June 2024

NTNU
Norwegian University of
Science and Technology

Henrik Hammarstrøm

Autonomous Loading Operations with a Shipboard Knuckle-boom Crane

Master's thesis in Marine Technology
Supervisor: Eilif Pedersen
Co-supervisor: Tom Arne Pedersen
June 2024

Norwegian University of Science and Technology
Faculty of Engineering
Department of Marine Technology

NTNU
Norwegian University of
Science and Technology

Preface

This master's thesis serves as my final work in my masters degree in Marine Cybernetics at the *Institute of Marine Technology* (IMT) at the *Norwegian University of Science and Technology* (NTNU). The thesis includes a bond graph model and a control system which can transport and lower cargo automatically. Additionally the thesis gives a brief summary of the main aspects a loading operation, and what an autonomous system can entail.

The thesis has been challenging as neither modeling nor autonomy are trivial problems, and as such I have gotten much advice in both topics from my supervisors. I would therefore like to thank my supervisor professor Eilif Pedersen and co-supervisor Tom Arne Pedersen for all meetings and being available for questions. I also want to thank Phd candidate Magnus Steinstø who assisted me in the compilation of code needed to create the integrated simulation model.

Abstract

Loading operations are a largely manual process which poses a health hazard for involved crew. Increasing the degree of automation for the systems involved can allow for mitigation of health hazards, and an increase in efficiency. In order to contribute to an increase in automation for the case of ship to shore loading, the thesis presents a functional control system and a bond-graph model composed of a ship, crane, wire and spreader.

The ship in the integrated model is the Revolt vessel, which is a concept for a short distance autonomous cargo ship created by DNV AS. Additionally, the crane used in the bond-graph model has three degrees of freedom and was created by an earlier masters student. The crane is mounted in the middle of the deck of Revolt, and is connected to a simplified wire and spreader mechanism.

The integrated system is implemented in a modular manner, and can be easily integrated or changed in order to suit the given case investigated by the modeler. Additionally, the control system is supervised by a finite state machine which is able to toggle between different modes of action such as crane transportation, sway compensation and wire lowering.

Oppsummering

Dagens lasteoperasjoner er i stor grad en manuell prosess som utgjør en helserisiko for involvert mannskap. Med å øke graden av automasjon i hvert av systemene involvert i en lasteoperasjon, kan operasjonen optimaliseres og antall skader reduseres. For å kunne bidra med en økning av automasjon for lasteoperasjoner som foregår mellom havn og skip, presenterer oppgaven et fungerende kontroll system og en bond-graf modell bestående av et skip, kran, vaier og conteineråk.

Skipet i simuleringsmodellen er Revolt, som er et konsept for et kortdistanses autonomt lasteskip skapt av DNV AS. Videre er kranen brukt i oppgaven prosjektert av en tildigere masterstudent, og har tre frihetsgrader. Kranen er montert på midten av dekket på båten, og er koplet til en forenklet vaier og conteineråk.

Det integrerte systemet er implementert på modulært vis, og kan enkelt integreres med eksterne modeller. I tillegg, er kontrollsystemet styrt av en tilstandsmaskin med mulighet til å bytte mellom oppgaver i form av kran transport, svai kompensering, og senking av last.

Table of Contents

List of Figures

List of Tables

1 Introduction

1.1 Motivation and background

Ship to shore loading operations typically makes use of cranes to move cargo on and off a ship. On the port-side it is common to use gantry cranes, which are either rail mounted or rubber tired. In regards to shipboard cranes, these can come in different variants such as knuckle, static or telescopic boom configurations. The cranes are typically controlled by on site operators using either a control pendant or actuator stick.

A loading operation can be decomposed to four main parts. These can be summarized as:

- Connecting and preparing the cargo.
- Controlling the horizontal motions, lifting and slewing of the cargo.
- Lowering and placing the cargo safely
- Securing the cargo on either the ship or portside

In the phase concerned with cargo preparation, the type of cargo becomes important. Non-standarized cargo in the form of break bulk often require hoisting gear to be installed, and then manually attached to either hooks or gripping mechanisms connected to the crane in question. However, standardized cargo such as containers allow for greater ease in automating the loading operation, and can be carried by use of a spreader mechanism, which come equipped with twist-lock mechanisms shown in Figure 1 which attach to each corner of a container.

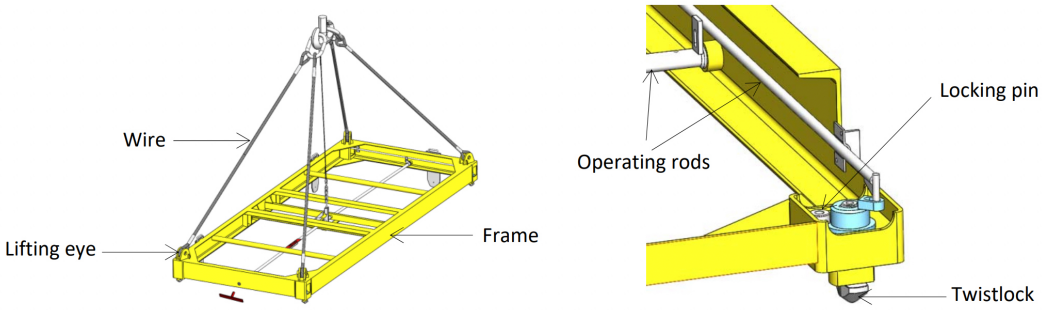


Figure 1: Illustration of spreader mechanism showing twist locks [1].

The phase concerned with lift, slew and horizontal transport have several challenges. Slew motions (horizontal rotations) of cargo often need direct crew involvement, where the crew or stevedores adjust the cargo during landing and pickup by the use of ropes or wires. The lifting and horizontal motions of the cargo is traditionally handled by the crane operator, and the movement of heavy cargo poses a safety risk to onboard personnel. The process of lowering the cargo is also crew intensive. The crane operator usually lowers the cargo manually, and if the operator loses visibility of the cargo the crew will assist by the use of hand signals and 2-way radios [2].

In regards to automation there exist automated container terminals, and according to [3] 3% of the worlds container terminals are either semi or fully automated. Where fully automated harbors

are able to automate both the horizontal as well as the vertical motions of the cranes, and semi-automated ports are only able to automate the vertical crane motions. Such ports often have cranes with automated systems in the form of collision avoidance, heave-compensation and anti-rotation systems on the cargo.

Smaller ports face constraints in securing adequate funding for near-term efficiency enhancements [2], and are often frequented by vessels relying on their own loading equipment. These vessels are often tasked with handling both containers and break bulk cargo. In order to contribute to an increase in automation for loading operations making use of a shipboard crane, the thesis presents a simulation model and control system containing an integrated ship, crane, wire and spreader model able to transport a spreader to a desired container in the world by utilizing a 3 degree of freedom (DOF) knuckle-boom crane.

1.2 Problem formulation

The thesis reviews the main particulars of a loading operation, and presents an integrated simulation model and a functional control system in order to perform automatic transport and lowering of cargo by using a shipboard knuckle-boom crane.

In order to create an integrated simulation model, the vessel parameters of Revolt had to be found. A hull model was therefore provided by DNV, and used to gain the necessary vessel parameters. Then the knuckle-boom crane presented in the master thesis of Gyberg [4] was then scaled by a factor of 6,5 in order to enable cargo transport to and from the ship. The crane was the attached in the middle of the deck of Revolt as this would provide the most stability. The spreader mechanism and wire was then modeled as a rigid body and an over-damped 3D spring respectively.

Once the integrated system was created the control system was implemented. The control system is supervised by a finite state machine which is able to toggle between different states. These states are crane transportation, sway-compensation and station-keeping of the crane tip, and finally wire lowering and heave compensation. In order to verify that the control system worked as intended, a series of case studies investigating each algorithm as well as the integrated system was performed.

1.3 Chapter overview

Chapter 2 ***Theory and background material*** describes the main particulars of the case the loading operation takes place in, and gives an overview of autonomous systems and bond graph modeling.

Chapter 3 ***Lagrangian and Hamiltonian mechanics*** introduces the concepts of reference frames, generalized and quasi coordinates. Additionally, the Lagrangian equations of motion is presented both by the use of generalized and quasi coordinates.

Chapter 4 ***Ship and crane dynamics*** describes the equations of motion of a ship, and describes the inertial and hydrodynamic terms of the equation. Additionally, the chapter describes the importance of geometric jacobians and the concept of inverse kinematics and workspace of an anthropomorphic manipulator.

Chapter 5 ***Ship and crane integration*** presents the equations used in integrating the ship and

crane, and how one can use a .dll in parallel with a *20-sim* application.

Chapter 6 ***Control*** gives a brief introduction to PID-controllers, and describes each of the control algorithm as well as the finite state machine used in the thesis.

Chapter 7 ***Modeling*** explains the assumptions and simplifications done to the models of the integrated system. Additionally, the most important bond graph models are presented.

Chapter 8 ***Results and discussion*** goes through a series of case studies in order to verify the implemented control algorithms, and presents the system performance of the integrated system under ideal conditions.

Chapter 9 ***Conclusion and further work*** comes with recommendations of further work and concludes the work performed in the thesis.

1.4 Previous and related work

In order to create the integrated simulation model as well as the control system several main sources have been used.

The master thesis of Gyberg [4] presented much of the necessary frame transformations and crane dynamics that was needed in order to create an integrated ship-crane model. In addition, the spreader and wire model developed in the project thesis [5] was also merged to the integrated model with minor modifications.

The integration of ship and crane by the use of quasi coordinates is heavily based on the paper *Modeling of Generic Offshore Vessel in Crane Operations With Focus on Strong Rigid Body Connections* [6], which describes the modeling of a generic ship and crane system. This paper illustrates how the quasi-coordinate equations of motion in state space form can be developed while using the body-fixed frame of the vessel as a common frame of reference. Additionally the book of *System dynamics* [7] and the lecture notes in *TMR4275 modeling and simulation of physical systems* [8] provided the necessary modeling theory in order to create the final bond graph model.

To gain a overview of loading operations the paper *Gap analysis for automated cargo handling operations with geared vessels frequenting small sized ports* [2] was used. The paper describes the main phases of a loading operations, as well as some of the current challenges that need to be solved in order to perform fully autonomous loading operations.

In order to develop equations of motion and perform crane control the books *Methods of analytical dynamics* [9] and *Modeling and Control of Robot Manipulators* [10], was heavily used. The book [9] introduces the use of quasi-coordinates as well as how one can express the Lagrangian equations of motion by utilizing quasi-coordinates. Additionally, [10] presents several control schemes and the importance of geometric jacobians in order to control various types of manipulators.

2 Theory and background material

2.1 Case considerations

One of the cases to be considered in the thesis, is the transport and placement of spreader from the ship to a container at port.

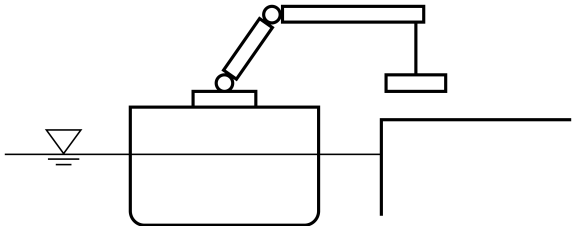


Figure 2: 2D sketch of shipboard crane performing a loading operation by harbor.

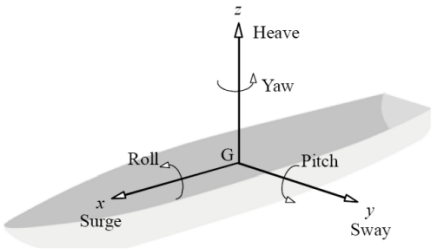


Figure 3: 6 DOFs of a vessel [11]

When investigating the case of ship to shore loading, it is worth considering the behaviors of the ship, crane and spreader mechanism. Regarding the dynamics of a ship, it is important to know that a ship is able to move in 6 degrees of freedom (DOF) as shown in Figure 3. The translational movements in x, y and z direction are named surge, sway and heave. Meanwhile, the rotations about each of these axis are usually represented as ϕ, θ and ψ and are called roll, pitch and yaw respectively.

There are inertial as well as hydrodynamic forces acting on a vessel. In the case with a ship by harbor one can assume a calm sea state, as such the wave frequencies of interest will be quite low, and can be approximated to be zero. This gives potential damping in many DOFs to be equal to zero, and makes viscous damping in the form of eddy making an important contributor to the ships damping. Particular DOFs of interest is roll, pitch and heave. This is due to the fact that the ship has only small movements in surge, sway and yaw while docked when assuming a calm sea state and negligible contributions from current and wind forces.

Furthermore, an important factor impacting the roll, pitch and heave motions is the hydrostatic restoring forces. The restoring forces in all three DOFs are due to the pressure the surrounding water exerts on the hull as a result of the displaced water volume. In still waters the heave motion of the ship is decided primarily by the hydrostatic restoring forces in z-direction and the weight from the ship including any loaded cargo or equipment. As for roll and pitch, one usually investigates the restoring moments arm GZ, where GZ is the moment arm between the center of gravity G, and the buoyancy force F_b.

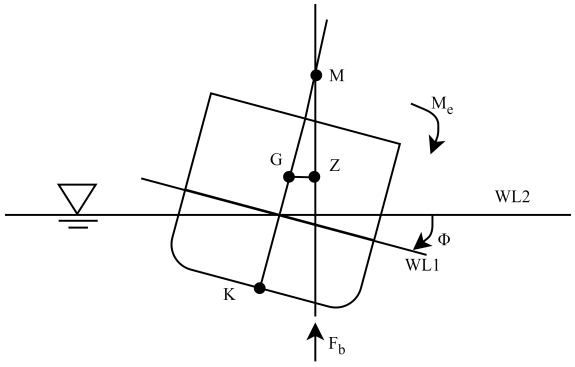


Figure 4: Principle sketch in roll. Inspiration taken from [12].

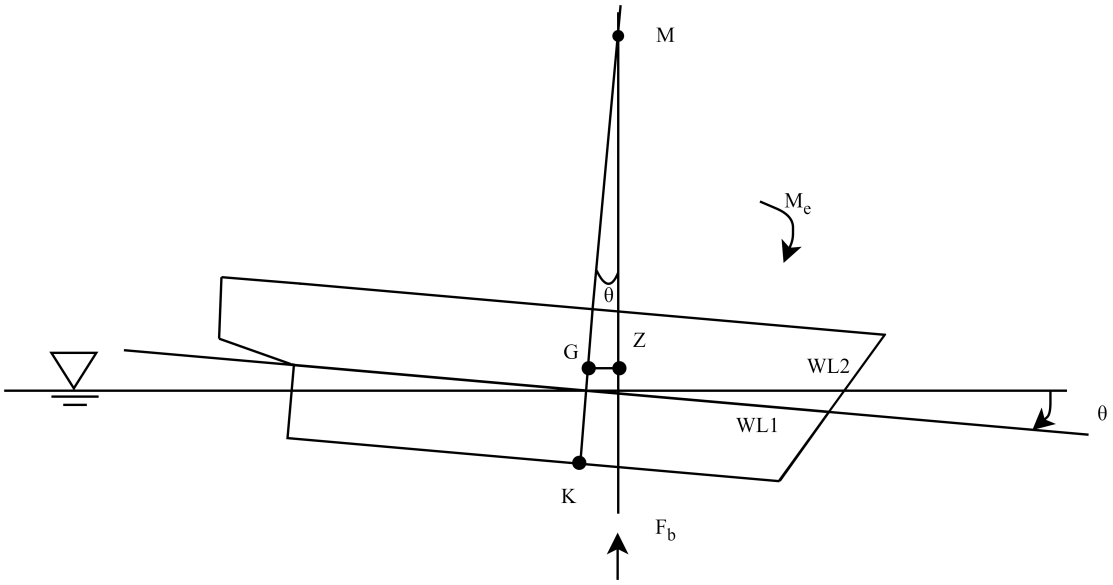


Figure 5: Principle sketch in pitch. Inspiration taken from [12].

The principal sketches Figure 4 and Figure 5, show the restoring arm GZ for a given roll and pitch angle as a result of an external trimming moment M_e. The sketches also show the positions of the keel K, center of gravity G and the metacenter M. Where the metacentre is the point of intersection of the buoyancy attack angle and the line spanning between the keel and center of gravity as shown in Figure 4 and Figure 5. In addition, the waterlines before and after rotation is denoted as $WL1$ and $WL2$, where $WL2$ is the waterline after rotation. Lastly, the buoyancy force F_b and its attack angle is also shown. The reason why the moment arm GZ is of particular interest, is that one can investigate for what rotation angles a ship will be able right itself up, and avoid capsizing. If for any reason the GZ value becomes less than zero the ship will capsize [12], which is a possibility when a ship is performing a loading operation.

In addition to the Revolt vessel from DNV, the thesis makes use of the knuckle-boom crane developed by Fredrik Gyberg [4]. This crane has 3 DOFs, and is able to rotate about the base, and each of its links. The crane was chosen as it is a generic offshore crane, and contains model parameters which allowed it to be scaled to the appropriate size to be able to perform a loading operation. Important behavior to note is the effect that the crane has on the stability on the ship, and how the crane movements effect the spreader and ship movements.

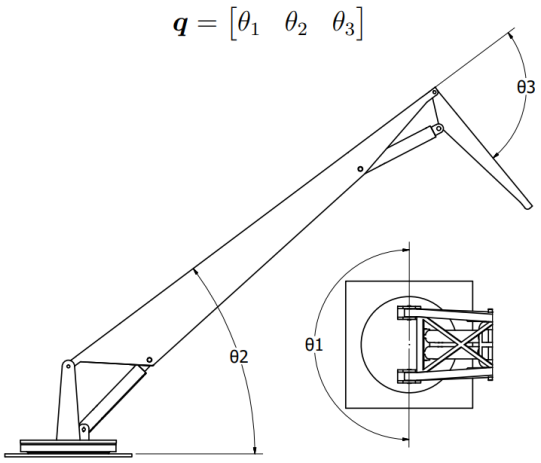


Figure 6: Crane from the side to the left. Base of the crane shown on the right [4]

The crane shown in Figure 6 shows the how each of the joints can rotate. Here rotation about the base is shown as θ_1 and rotation about the first and second joint is denoted as θ_2 and θ_3 respectively. In addition, the spreader which is attached to the crane tip through a wire is able to move in 4 DOF. These DOF consist of translational displacements along the x, y and z axis, and the rotation about the z axis. As the case contains no environmental forces, the pendulum motions induced on the spreader comes from transporting the spreader to a desired position. The vertical motions will come due to the rotation of the ship, but also due to wire lowering. An important simplification in the loading operation is that the rotation of the spreader mechanism is neglected, as controlling and modeling necessary tugger wires is beyond the scope of the thesis.

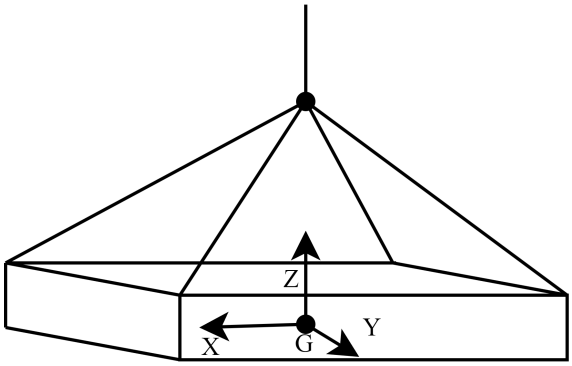


Figure 7: Principal sketch of spreader

The sketch in Figure 7 shows the body-fixed frame and how the wires are connected to each end of the spreader. It is important to note that the wires prevent rotational motion about the x, and y axis, and that twist lock mechanisms have been excluded from the sketch. This was done as the thesis will not be investigating the twist lock mechanics when the spreader attaches to a container. Instead the spreader will be defined as attached to the container if it is in close enough proximity to the container in question. Since the thesis does not allow the spreader to rotate, and the container is assumed to be the same orientation as the spreader. The orientation of the spreader is not checked before defining the spreader and container as attached.

2.2 An introduction to autonomous systems

As the field of robotics have become more advanced with increases of computational power and the recent advances in artificial intelligence, scientists and engineers are looking to increase the degree of autonomy in both the automotive and maritime fields, with the end goal of making cars and ships autonomous. The problem of autonomy can be divided into three main parts, which can be summarized to the sense-plan-act methodology discussed in [13].

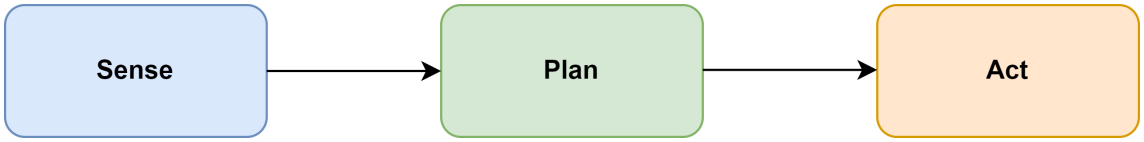


Figure 8: Sense plan act methodology

In Figure 8, **sense** is concerned with utilizing sensors such as camera, lidar or sonar in order to map the environment. **Plan** consists of the ability to use sensor data in order to make decisions such as stopping or generating new tracks to follow. Lastly, **act** is the robots ability to physically accomplish the control objective by utilizing its actuators. These three key objectives can be seen in more advanced autonomous system architectures, but it is worth noting that depending on the type of problem to be solved autonomously, the system architecture may change.

A proposed architecture for autonomous loading using the **Sense**, **Plan** and **Act** methodology could involve the following modules. The system would have a perception module able to detect the cargo by use of machine vision and markers. The system would then have a planning module, which based on the data fed by the perception module, would generate an appropriate route of the crane in order to avoid collision. Finally the system would utilize a control module to automatically attach to the cargo, and then transport it while compensation for heave, sway and heave motions while an external operator supervises the loading operation.

Another example of an autonomy architecture can be found in [14] where autonomous underwater vehicles (AUVs) is investigated. The book [14] presents an architecture consisting of planning and decision making, sensing and perception, monitoring and diagnosis, learning and adaptation as well as networking and collaboration. The modules monitoring and diagnosis and sensing and perception utilize both intereo as well as extereoceptive sensors in order to give the AUV situational awareness of both its internal conditions, as well as map the surrounding environment. The gathered data is then utilized in the planning and decision making module in order to choose the optimal behavior to accomplish the AUVs goal. Learning and adaptation is the capability to adapt to unexpected events with no a priori information or knowledge [14]. Lastly networking and cooperation, is the capability of multiple drones working together in order to accomplish a goal.

Regardless of the task to be solved, an autonomous system requires a way to toggle between different behaviors. These behaviors are a collection of actions or maneuvers that allow the system in question to accomplish its goal [14]. Examples of a behaviors that could be toggled for an AUV is docking, altitude keeping or seafloor-mapping. A solution to behavior toggling can be found in the subsumption architecture presented in [15]. This architecture was created with an autonomous mobile robot in mind, and divides the problem of creating a autonomous mobile robot into what is described as levels of competence. These range from avoiding contact with both moving and stationary objects at the lowest level, to reasoning about the behavior of objects at the highest

level [15]. These levels of competence are achieved by creating different layers of control, where each control layer is able to interface with with the layer below it. Each of the control layers can work on their own individual goals, but is kept in check by the use of a suppression mechanism that mediates the actions taken. This allows for multiple goals being worked towards at the same time, but allow certain goals such as collision avoidance to take precedence when needed.

Another method of choosing behaviors instead of using a suppression mechanism, is to organize each behavior into states and utilize a finite state machine (FSM) in order to toggle between them. An FSM consists of a set of states, and a set of transitions between pairs of states [16]. A transition takes in a condition that causes the transition in state, and performs an action during the state transition. In order to display FSMs, one can utilize state diagrams. Here states are shown as circles, and transitions are shown as arrows from a source to a target state. Each arrow is labeled with the condition and action of the transition [16].

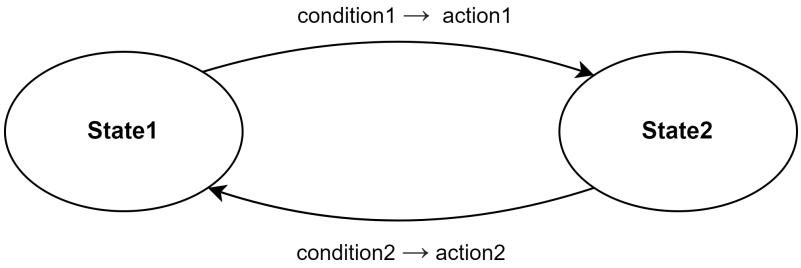


Figure 9: State diagram showing two states and two transitions

In Figure 9 an illustration of how a state diagram can be drawn is shown. The state machine is composed of ***State1*** and ***State2***, and contains two transitions. To each transition there is a condition and a action that will trigger should the condition be met.

Utilizing an FSM approach in order to change the behavior of a robot is suited for pre-scripted mission plans, where the FSM can either toggle between states based on what task has been accomplished, or change states based on the amount of time has passed since the last change of state. In order to account for uncertainties in the operation it is possible to add fallback states, which are states that sets the autonomous system into a minimal risk condition. It is worth noting however that these fallback states need to be implemented ahead of time as well, and will thus only catch situations that the programmer has deemed likely to occur during the operation.

2.3 Bond graph modeling

The following section is based on [8] and [7] and aims to give a brief summary of the bond graph language, and how it can be used to model dynamical systems. For a more in depth explanation of all constitutive laws the reader is advised to read [8] or [7] where the bond-graph modeling language is thoroughly presented.

The bond graph modeling language allows for modular system implementations in a graphical manner, and considers the energy flow, storage and dissipation between system components. The modeling language considers a system to be composed of one or more sub-models, which can be further simplified to elements. Energy is transferred instantaneously and without loss between components by the use of ports and power bonds as can be seen in Figure 10. Elements with a single port are called single-port elements, and elements with multiple ports are categorized as multi-port elements.

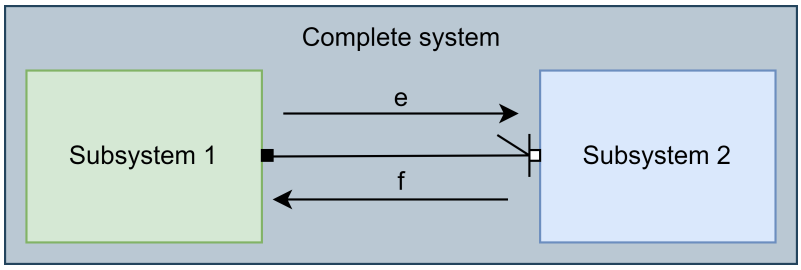


Figure 10: A larger system consisting of 2 subsystems connected with a power bond. Ports are shown as a black and white square.

Bond graph modeling utilizes four different variables in order to describe the energy flow of dynamic systems. These are the power variables effort *e(t)* and flow *f(t)* as well as the generalized displacement *q(t)* and generalized momentum *p(t)*. The power variables *e(t)* and *f(t)* are transferred by the use of a power bond, where the half-arrow indicates the positive power direction. The power P delivered from a power bond can be defined in terms of the effort and flow variable as

$$P = e \cdot f. \tag{1}$$

Additionally the generalized momentum and displacement are defined from the following constitutive relations [8]:

$$p(t) = \int_0^t e(t)dt + p(0) \tag{2}$$

$$q(t) = \int_0^t f(t)dt + q(0) \tag{3}$$

The physical meaning behind the flow, effort and generalized momentum and displacement differs depending on the domain of the system being modeled. In the case of a mechanical system concerned with translations, the effort and flow variable will denote force $[N]$ and translational velocity $[m/s]$ respectively. The generalized displacement and momentum will then denote distance $[m]$

and the linear momentum $[kgm/s]$. In the case of a mechanical system concerned with rotation the effort and flow will be moments $[Nm]$ and angular velocities $[rad/s]$. The generalized momentum and displacement then becomes angular momentum $[Nms]$ and angles $[rad]$.

In order to dissipate, provide, transform as well as store both kinetic and potential energy, several single and multi-port elements are used. Energy dissipation can be modeled by the use of the single-port R-element, which can represent friction in a mechanical system. The single-port source elements Sf and Se can introduce energy to a system by setting either a flow or an effort respectively. Transformation of energy can be represented by the two-port transformer element TF, and storage of kinetic energy can be modeled by the single-port I-element which can represent inertia for a mechanical system. Lastly the storage of potential energy can be modeled as the single-port C-element, which can represent a mechanical spring.

To connect the single and multi-port elements to a larger bond graph, one can make use of junctions. The two main junctions are the common effort junction *0-junction* and the common flow junction *1-junction*. Each of these have their own laws. Inspecting Figure 11a showing the numbered 1-junction the following relations holds

$$e_1 - e_2 - e_3 = 0 \tag{4}$$

$$f_1 = f_2 = f_3. \tag{5}$$

For the numbered 0-junction in Figure 11b the opposite holds

$$e_1 = e_2 = e_3 \tag{6}$$

$$f_1 - f_2 - f_3 = 0. \tag{7}$$

(a) 1-junction with causal strokes

(b) 0-junction with causal strokes

Figure 11: The common flow and effort junctions with assigned causality and numbered power bonds.

In addition to transferring power, power-bonds also show the causality by the use of the causal stroke which is represented as the vertical line on the power bond. The direction of the causal stroke determines whether an effort is fed into or comes out of a given port element, as can be seen in Figure 10. The junctions as well as several of the elementary bond graph elements have causal considerations. The common effort junction can only have a single power bond providing the effort, while the common flow junction can only have a single power bond providing the flow to the junction as seen in Figure 11. Both of the source elements have a fixed causality, with the flow source having fixed flow out, while the effort source having fixed effort out. The R-element can either have a flow or effort entering the element, but the I and C element can have unwanted

causality configurations called differential causality. Differential causality is a result of having to differentiate instead of integrating in order to provide an effort or flow, and is unwanted due to mathematical reasons.

Elements	**Constitutive laws**
S_e ⟶	e
S_f ⟶	f
MS_e ⟶	$e(t)$
MS_f ⟶	$f(t)$
⟶ R	$e = Rf$ (linear) $e = \phi_R(f)$ (non-linear)
⟶ C	$q = Ce$ (linear) $q = \phi_C(e)$ (non-linear)
⟶ I	$p = If$ (linear) $p = \phi_I(f)$ (non-linear)
$\overset{1}{\longrightarrow} TF \overset{2}{\longrightarrow}$	$e_1 = me_2$, $f_2 = mf_1$
$\overset{1}{\longrightarrow} MTF \overset{2}{\longrightarrow}$	$e_1 = m(t)e_2$, $f_2 = m(t)f_1$

Table 1: Constitutive laws for common bond graph elements used in the thesis. The laws can be found in [8] and [7].

Table 1 shows the constitutive laws for the most commonly used bond graph elements used in the thesis. Here I,C and R are the constant inertia, compliance and resistance parameters, while $\phi_I(f), \phi_C(e)$, $\phi_R(f)$ are nonlinear functions. The parameters m and $m(t)$ are the constant and time-varying modoluses. For a more comprehensive overview on how the laws change with regards to causality, the reader is recommended to read [7].

2.4 Fields and vectorial power bonds

When working with systems consisting of multiple DOFs it can be useful to use vectors and matrices instead of scalars as this will allow for more compact bond-graphs. Similarly to the scalar power bonds described in Section 2.3, vectorial power bonds also provide power in the direction of the direction of the half-arrow.

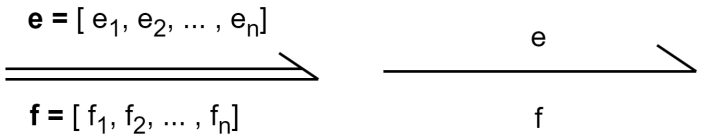


Figure 12: Vectorial and scalar power bonds.

However now the effort and flow variables associated with the power bond becomes vectors as seen in Figure 12:

$$\mathbf{f(t)} = [f_1(t), f_2(t), \ldots, f_n(t)]^T \tag{8}$$

$$\mathbf{e(t)} = [e_1(t), e_2(t), \ldots, e_n(t)]^T \tag{9}$$

Additionally the power provided from the vectorial power bond becomes:

$$\mathbf{P}(\mathbf{t}) = \mathbf{e}(\mathbf{t}) \cdot \mathbf{f}(\mathbf{t}). \tag{10}$$

The earlier single port elements are now changed to take in vectors rather than scalars and R, C, I-elements now become fields containing several scalar ports. Additionally the IC-field is introduced. This field has the characteristics of both an I and C elements, and can allow for more compact bond graphs of rigid bodies while avoiding differential causality.

2.5 Power muxer and demuxer

An important element that the thesis makes use of is the power muxer and demuxer, as the integrated model uses power bonds of different dimensions depending on the submodel.

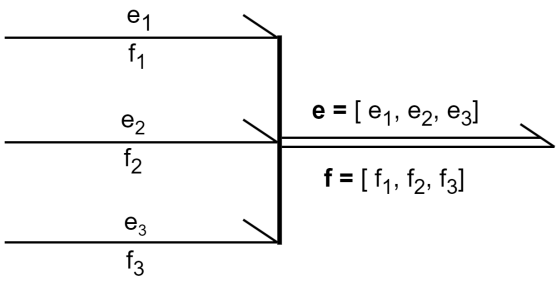


Figure 13: Illustration of power muxer operating in 3 dimensions

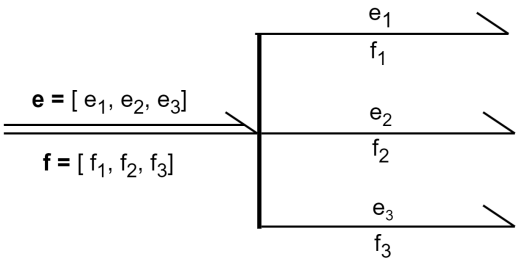


Figure 14: Illustration of power demuxer operating in 3 dimensions

As seen in Figure 13 a power muxer is able to compose a power bond containing each of the scalar bond graphs. The power demuxer shown in Figure 14 shows how a three dimensional vectorial power bond can be decomposed into three separate power bonds. The power muxer and demuxer is capable of performing these operations with n dimensional input and output power bonds.

3 Lagrangian and Hamiltonian mechanics

The following section is based on the theory presented in [9] and will introduce the concepts of coordinate frames, as well as quasi- and generalized coordinates. Then the concepts of potential, as well as kinetic energy will be presented. Lastly, the Lagrangian equation of motion presented with both generalized and quasi-coordinates will be shown.

3.1 Reference frames

In order to describe the movement of a rigid body moving in $\mathbb{R}^3$, one needs to be familiar with the concept of reference frames. A reference frame is its own coordinate system and can be defined by its linear independent base vectors $\mathbf{i}$, $\mathbf{j}$, $\mathbf{k}$, and its origin $p_O = [x_O, y_O, z_O]^T$ from which the base vectors originate. Typically one defines these base vectors to be orthonormal, and in accordance with the right hand rule. It is worth noting that any vector $\mathbf{r}$ can be resolved in one or more coordinate systems [9].

When working with more than one reference frame, it is useful to add a superscript and subscript of vectors and points in order to explain which coordinate frame the point or vector is explained in terms of. In the case of two different frames $\{0\}$ and $\{1\}$, then the vector $\mathbf{r^0_{1/0}}$ is described in terms of the $\{0\}$ frame as shown in the superscript. The subscript $\{1/0\}$ explains that the origin of the vector starts in $\{0\}$ and spans to $\{1\}$. Utilizing the aforementioned conventions, one can easily keep track of how one frame moves in relation to another, which is useful when one describes the movement of reference frames relative to a inertial or common reference frame.

A useful tool for transforming from one coordinate system in $\mathbb{R}^3$ to another is utilizing rotation matrices. The elementary rotation matrices allow for rotating along the x, y and z axis. Following the conventions presented in [17] they are defined as $\mathbf{R}_{x,\phi}$, $\mathbf{R}_{y,\theta}$ and $\mathbf{R}_{z,\psi}$ respectively, where the rotation angle about the x, y and z axis is defined as ϕ, θ and ψ.

$$\mathbf{R}_{x,\phi} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix}, \mathbf{R}_{y,\theta} = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix}, \mathbf{R}_{z,\psi} = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{11}$$

The rotation matrices have several key properties, such as not being commutative [9] $\mathbf{R_x\ R_y\ R_z} \neq \mathbf{R_z\ R_y\ R_x}$, and being orthogonal which implies $\mathbf{R^{-1} = R^T}$.

3.2 Generalized coordinates

Generalized coordinates can be defined as the set of states that are able to express all possible system configurations. For a system with n degrees of freedom, the generalized coordinates of a system is represented in vectorial form as [9]

$$\mathbf{q_k} = [q_1, q_2, \ldots, q_n]^T. \tag{12}$$

It is important to note that the choice of generalized coordinates are not unique, and need to at least correspond to the amount of degrees of freedom the system has.

3.3 Quasi coordinates

Quasi-coordinates are defined as linear combinations of the time rates of the generalized coordinates of a system, where the generalized coordinate ω_s can be presented as [9]

$$\omega_s = \alpha_{1s}\dot{q}_1 + \alpha_{2s}\dot{q}_2 + \cdots + \alpha_{ns}\dot{q}_n \tag{13}$$

In vectorial form this becomes:

$$\boldsymbol{\omega} = \boldsymbol{\alpha}^{\boldsymbol{T}}\dot{\mathbf{q}}_{\mathbf{k}} \tag{14}$$

Here $\boldsymbol{\omega}$ is the vector representing each quasi coordinate. $\boldsymbol{\alpha}^{\boldsymbol{T}}$ is the transposed $n \times n$ transformation matrix, and $\dot{\mathbf{q}}_{\mathbf{k}}$ is the vector containing the timerate of each generalized coordinate.

3.4 The Lagrangian equation of motion

In order to present the Lagrangian, expressions for the kinetic end potential energy should be presented. The kinetic energy of a rigid body existing in $\mathbb{R}^3$ is given as a function of both angular and translational velocity:

$$T = \frac{1}{2}m\mathbf{v}_{\mathbf{c}}^{\mathbf{T}}\mathbf{v}_{\mathbf{c}} + \frac{1}{2}\boldsymbol{\omega}^{\boldsymbol{T}}\mathbf{I}\boldsymbol{\omega}. \tag{15}$$

Here m is the mass of the rigid body, and $\mathbf{v_c}$ is the translational velocity vector of the center of mass in the rigid body. Meanwhile, $\boldsymbol{\omega}$ and $\mathbf{I}$ are the rotational velocity vector and rotational inertia matrix of the body [9].

The expression of potential energy of a rigid body can take different forms based on the presence of restoring or gravitational forces acting on the body. However the potential energy will not be a function of the velocities, but rather the position vector $\mathbf{r}$ of the system [9]. A general expression of the potential energy V can be given as:

$$V(\mathbf{r}) = \int_{\mathbf{r}}^{\mathbf{r_0}} \mathbf{F}d\mathbf{r}. \tag{16}$$

Where $\mathbf{F}$ is a given force acting on the system, and $\mathbf{r_0}$ and $\mathbf{r}$ is a reference position and system position respectively [9].

Now the Lagrangian can be defined as the difference in kinetic and potential energy as:

$$L = T - V. \tag{17}$$

Utilizing generalized coordinates, the Lagrangian equation of motion is found as [9]:

$$\frac{d}{dt}\frac{\partial L}{\partial \dot{q}_k} - \frac{\partial L}{\partial q_k} = Q_k. \tag{18}$$

Here Q_k represents the generalized forces acting on the system, and q_k represents the k-th generalized coordinate. Another way of representing the same equation is represented in [7]

$$\frac{d}{dt}\frac{\partial T}{\partial \dot{q}_k} - \frac{\partial T}{\partial q_k} = E_k. \tag{19}$$

where E_k contains both generalized forces, as well as terms from the potential energy V. Utilizing Equation 19, it is possible to rewrite the equation into Hamiltonian form. This can be done as shown in [7] by first defining the generalized momentum p_k as

$$p_k = \frac{\partial T}{\partial \dot{q}_k}. \tag{20}$$

Equation 19 can now be rewritten into:

$$\dot{p}_k = \frac{\partial T}{\partial \dot{q}_k} + E_k. \tag{21}$$

In matrix form this becomes:

$$\mathbf{p_k} = \mathbf{M}(\mathbf{q_k}, t)\dot{\mathbf{q}}_k + \mathbf{a}(\mathbf{q_k}, t). \tag{22}$$

Here $\mathbf{M}(\mathbf{q_k}, t)$ is a symmetric matrix, and $\mathbf{a}(\mathbf{q_k}, t)$ is a vector that occurs when the system includes time-varying velocity sources as explained in [7]. Finally solving for $\dot{q}_k$ this results in:

$$\dot{\mathbf{q}}_\mathbf{k} = \mathbf{M}(\mathbf{q_k}, t)^{-1}[\mathbf{p} - \mathbf{a}(\mathbf{q_k}, t)]. \tag{23}$$

3.5 Lagrangian equation of motion in quasi coordinates

When utilizing quasi coordinates one needs to take into account the transformation matrices between generalized and quasi coordinates when developing the equations of motion. The Lagrangian equation of motion found in Equation 18 then turns into:

$$\frac{d}{dt}\frac{\partial \bar{T}}{\partial \boldsymbol{\omega}} + \boldsymbol{\beta}^T\boldsymbol{\gamma}\frac{\partial \bar{T}}{\partial \boldsymbol{\omega}} - \boldsymbol{\beta}^T\frac{\partial \bar{T}}{\partial \mathbf{q}} + \boldsymbol{\beta}^T\frac{\partial V}{\partial \mathbf{q}} = \boldsymbol{\beta}^T Q_k. \tag{24}$$

Here $\bar{T}$ is the kinetic energy as a function of generalized and quasi coordinates as shown in [9]. Additionally, the transformation matrix $\boldsymbol{\beta}$ is defined as: $\boldsymbol{\beta} = \left(\boldsymbol{\alpha}^{\boldsymbol{T}}\right)^{-1}$, and relates the time rates of the generalized coordinates to the quasi coordinates. Additionally γ is defined as the n $\times$ n matrix:

$$\gamma = \begin{bmatrix} \xi_{11} & \dots & \xi_{1n} \\ \vdots & \ddots & \vdots \\ \xi_{n1} & \dots & \xi_{nn} \end{bmatrix} - \begin{bmatrix} \boldsymbol{\omega}^T \boldsymbol{\beta}^T \frac{\partial \boldsymbol{\alpha}}{\partial q_1} \\ \vdots \\ \boldsymbol{\omega}^T \boldsymbol{\beta}^T \frac{\partial \boldsymbol{\alpha}}{\partial q_n} \end{bmatrix} \tag{25}$$

$$\xi_{ij} = \boldsymbol{\omega}^T \boldsymbol{\beta}^T \frac{\partial \alpha_{ij}}{\partial q_n} \tag{26}$$

Here $\frac{\partial \alpha_{ij}}{\partial q_n}$ is the element-wise partial derivative of the alpha matrix with respect to the n-th generalized coordinate. It is important to note that the choice of using quasi-coordinates or generalized coordinates is up to the preference of the modeler. Generalized coordinates makes it so that the kinetic energy of the system needs to be expressed in regards to the inertial frame. However, modeling a system using quasi-coordinates can allow the kinetic energy to be expressed in terms of a non-inertial frame at the cost of using the aforementioned transformation matrices.

4 Ship and crane dynamics

The following chapter uses the theory presented in [17] and [18] in order describe the equations of motion for a ship. Then the theory presented in [10] will be used to review the concept of geometric jacobians as well as inverse kinematics for an anthropomorphic manipulator.

4.1 Equation of motion of a ship in 6 DOF

As mentioned in Section 2.1 a ship is able to move in 6 DOF. These DOFs are translational movement along the x, y and z axes, as well as the rotations along each axis. In order to describe the motion of the body-fixed reference frame {b} relative to an inertial frame {0} the following equation can be utilized:

$$(\mathbf{M}+\mathbf{A})\dot{\mathbf{v}}^{\mathbf{0}}_{\mathbf{b/0}}+\mathbf{C}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}+\mathbf{D}\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}+\mathbf{G}\eta^{\mathbf{0}}_{\mathbf{b/0}}=\boldsymbol{\tau}^{\mathbf{0}}_{\mathbf{b/0}}. \tag{27}$$

The term $(\mathbf{M}+\mathbf{A})\dot{\mathbf{v}}^{\mathbf{0}}_{\mathbf{b/0}}$ represents the inertial forces. Here $\mathbf{A}$ is the 6×6 added mass matrix and $\mathbf{M}$ is the 6×6 inertia tensor denoted as [17] and also presented in [5],

$$M=\begin{bmatrix} m & 0 & 0 & 0 & mz_G & -my_G \\ 0 & m & 0 & -mz_G & 0 & mx_G \\ 0 & 0 & m & my_G & -mx_G & 0 \\ 0 & -mz_G & my_G & I_x & -I_{xy} & -I_{xz} \\ mz_G & 0 & -mx_G & -I_{yx} & I_y & -I_{xz} \\ -my_G & mx_G & 0 & -I_{zx} & -I_{zy} & I_z \end{bmatrix} \tag{28}$$

where x_G, y_G, z_G represents the x, y and z coordinate of the center of gravity of the vessel. In addition, I_x, I_y, I_z is the rotational inertia of the vessel about the x, y and z axis respectively, and m is the mass of the vessel. Lastly, I_{zx}, I_{zy}, I_{xy}, I_{xz}, I_{yx} and I_{xz} are known as the products of inertia, and describes the imbalance of mass distribution of the rigid body in question.

Additionally the coriolies and centripetal matrix $\mathbf{C}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}$ is composed of contributions from the hydrodynamic added mass and rigid body centripetal matrices as:

$$\mathbf{C}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}=\mathbf{C}_{\mathbf{RB}}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}+\mathbf{C}_{\mathbf{A}}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}} \tag{29}$$

The term $\mathbf{C}_{\mathbf{A}}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}$ represent the hydrodynamic coriolis and centripetal forces, while $\mathbf{C}_{\mathbf{RB}}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}}$ describes the rigid body coriolies and centripetal forces acting on the vessel. $\mathbf{C}_{\mathbf{RB}}(\mathbf{v}^{\mathbf{0}}_{\mathbf{b/0}})$ is a 6×6 matrix as described in [17] and also presented in [5]:

$$C_{RB}(v) = \begin{bmatrix} 0 & 0 & 0 & c_{41} & -c_{51} & -c_{61} \\ 0 & 0 & 0 & c_{42} & c_{52} & -c_{62} \\ 0 & 0 & 0 & -c_{43} & -c_{53} & c_{63} \\ -c_{41} & -c_{42} & c_{43} & 0 & -c_{54} & -c_{64} \\ c_{51} & -c_{52} & c_{53} & c_{54} & 0 & -c_{65} \\ c_{61} & c_{62} & -c_{63} & c_{64} & c_{65} & 0 \end{bmatrix} \tag{30}$$

$$\begin{aligned} c_{41} &= mz_G r & c_{42} &= m\omega & c_{43} &= m(z_G p - v) \\ c_{51} &= m(x_G q - \omega) & c_{52} &= m(z_G r + x_G p) & c_{53} &= m(z_G q + u) \\ c_{54} &= I_{zx} p - Ixr & c_{61} &= m(v + x_G r) & c_{62} &= -mu \\ c_{63} &= mx_G p & c_{64} &= I_y q & c_{65} &= I_x p + I_{xz} r \end{aligned}$$

4.2 Hydrodynamic terms

In addition to inertial and centripetal terms, the equation of motion also contain the added mass matrix $\mathbf{A}$, and hydrodynamic damping and restoring forces from the terms $\mathbf{Dv^0_{b/0}}$ and $\mathbf{G}\eta^{\mathbf{0}}_{\mathbf{b/0}}$. Added mass can be viewed as a virtual mass an accelerating body displaces as the body travels through a fluid, as the body and fluid cannot occupy the same space at the same time [17]. The damping matrix $\mathbf{D}$ can be viewed as a function of both potential and viscous damping, where viscous damping can either be approximated from empirical formulas or found from experiments. Estimates of the potential damping can be found by investigating the classical frequency-domain model. By exciting the frequency-domain model with an external force at different frequencies it is possible to gain approximations for the potential wave radiation damping, as well as develop expressions for added mass. However, when observing the case of a calm sea state, the excitation frequency can be approximated to zero, and as such the added mass, and damping coefficients only need to be found for a specific frequency. The damping matrix can be expressed as [17]

$$\mathbf{D}(\omega)_{\mathbf{tot}} = \mathbf{D_{pot}}(\omega) + \mathbf{D_v}(\omega). \tag{31}$$

Where $\mathbf{D_{pot}}(\omega)$ is the frequency dependent potential wave radiation damping, and $\mathbf{D_v}(\omega)$ is the frequency dependent viscous damping, both being 6×6 matrices. Meanwhile the restoring force matrix G can be defined as [18],[5]

$$C_{33} = \rho g A_{wp} \tag{32}$$

$$C_{35} = C_{53} = -\rho g \int\int_{A_{wp}} x ds \tag{33}$$

$$C_{44} = \rho g \nabla (z_B - z_G) + \rho g \int\int_{A_{wp}} y^2 ds = \rho g \nabla \overline{GM}_T \tag{34}$$

$$C_{55} = \rho g \nabla (z_B - z_G) + \rho g \int\int_{A_{wp}} x^2 ds = \rho g \nabla \overline{GM}_L. \tag{35}$$

When linearizing and assuming that the ship in question has yz-symmetry the resulting restoring force matrix becomes

$$G = \begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & \rho g A_{wp} & 0 & 0 & 0 \\ 0 & 0 & 0 & \rho g \nabla \overline{GM}_T & 0 & 0 \\ 0 & 0 & 0 & 0 & \rho g \nabla \overline{GM}_L & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}. \tag{36}$$

Here $\overline{GM}_T$ and $\overline{GM}_L$ are the GM value in roll and pitch respectively. Meanwhile ρ, g and ∇ represent the fluid density, gravitational acceleration and displaced water volume. A_{wp} is the waterline area of the vessel. Additionally, the hydrodynamic coriloies and centripetal matrix $\mathbf{C_A}(\mathbf{v})$ is defined according to the maneuvering theory presented in [17] as:

$$\mathbf{C_A}(\mathbf{v}) = \begin{bmatrix} \mathbf{0}_{3\times 3} & -\mathbf{S}(\mathbf{A_{11}v_1} + \mathbf{A_{12}v_2}) \\ -\mathbf{S}(\mathbf{A_{11}v_1} + \mathbf{A_{12}v_2}) & -\mathbf{S}(\mathbf{A_{21}v_1} + \mathbf{A_{22}v_2}) \end{bmatrix} \tag{37}$$

Here v_1 and v_2 are the translational and angular velocity vectors respectively. $\mathbf{S}(\mathbf{x})$ is the cross-product operator, and $\mathbf{A_{11}}$, $\mathbf{A_{12}}$, $\mathbf{A_{21}}$ and $\mathbf{A_{22}}$ are 3×3 matrices that make up the system inertia matrix of added mass as [17]:

$$\mathbf{A} = \begin{bmatrix} \mathbf{A_{11}} & \mathbf{A_{12}} \\ \mathbf{A_{21}} & \mathbf{A_{22}} \end{bmatrix} \tag{38}$$

4.3 Crane dynamics

A crane can be viewed as an anthropomorphic robotic manipulator, and can be composed of one or more links that are connected to either revolute or prismatic joints. Revolute joints induce rotational motion, while prismatic joints induce translational motion.

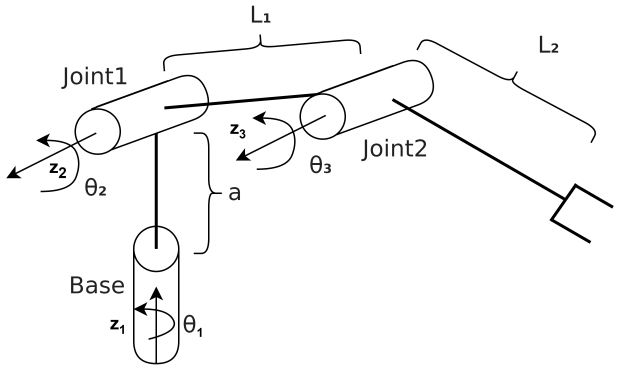


Figure 15: Sketch of anthropomorphic manipulator showing revolute joints and rotation axi. Inspiration taken from [10]

In order to describe the motion of a robotic manipulator, expressions for kinetic as well as potential energy of the system needs to be developed. This requires being able to describe the translational and angular velocity of the center of gravity of each link on the robotic manipulator for a common frame of reference. In order to accomplish this, describing the relationship between joint velocities and the angular and translational velocities of each center of mass by the use of geometric jacobians becomes important.

A geometric jacobian $\mathbf{J}$ is able to relate joint velocities $\dot{\mathbf{q}}$ to the translational and angular velocities at a given point as [10]

$$\mathbf{v} = \begin{bmatrix} \dot{\mathbf{p}} \\ \boldsymbol{\omega} \end{bmatrix} = \mathbf{J}(\mathbf{q})\dot{\mathbf{q}} \tag{39}$$

where $\dot{\mathbf{p}}$ and $\boldsymbol{\omega}$ are vectors representing the translational as well as angular velocity of the point. However, in order to find a geometric jacobian, it is worth reviewing how one can perform frame transformations. A frame transformation can be described by utilizing the elementary rotation matrices, where a frame transformation from a given inertial frame {0} to given frame {n} can be described as

$$\mathbf{R_0^n} = \mathbf{R_{n-1}^n} \dots \mathbf{R_1^2 R_0^1}. \tag{40}$$

Consequently the positional vector describing the position of frame {n} relative to {0} in terms of frame {0} can be transformed to describe the position relative to a given coordinate frame {i} by the use of transformation matrices as:

$$\mathbf{r_{n/0}^i} = \mathbf{R_0^i r_{n/0}^0} \tag{41}$$

Now consider the derivative of a rotation matrix which is defined as:

$$\dot{\mathbf{R}} = \mathbf{S}(\mathbf{t})\mathbf{R}(\mathbf{t}) \tag{42}$$

Where $\mathbf{S}(\mathbf{t})$ can have the following physical definition [10]:

$$\mathbf{S(t)} = \begin{bmatrix} 0 & -\omega_z & \omega_y \\ \omega_z & 0 & -\omega_x \\ -\omega_y & \omega_x & 0 \end{bmatrix}. \tag{43}$$

When observing a physical system ω_x, ω_y, ω_z can be interpreted as the angular velocities along the x,y and z axis of the rigid body whose body-fixed frame is the time varying reference frame $\mathbf{R(t)}$. Now the movement of the point $\mathbf{p(t)}$ in space described as a function of the time varying rotation matrix $\mathbf{R(t)}$ and the constant vector **p'** can be shown as [10]:

$$\mathbf{p(t)} = \mathbf{R(t)p'} \tag{44}$$

, which will have its velocity described as:

$$\dot{\mathbf{p}}(\mathbf{t}) = \dot{\mathbf{R}}(\mathbf{t})\mathbf{p}' = \mathbf{S(t)R(t)p}' = \boldsymbol{\omega(t)} \times \mathbf{R(t)p'} \tag{45}$$

Using these results the movement and velocity of a point in space relative to a given frame $\{n\}$ relative to a frame $\{0\}$ can be described as:

$$\mathbf{p^0_{n/0}} = \mathbf{0^0_{n/0}} + \mathbf{R^0_n p^n_{n/n}} \tag{46}$$

$$\dot{\mathbf{p}}^{\mathbf{0}}_{\mathbf{n/0}} = \dot{\mathbf{0}}^{\mathbf{0}}_{\mathbf{n/0}} + \boldsymbol{\omega(t)} \times \mathbf{R(t)}^{\mathbf{0}}_{\mathbf{n}} \mathbf{p}^{\mathbf{n}}_{\mathbf{n/n}} \tag{47}$$

This can be further simplified to:

$$\dot{\mathbf{p}}^{\mathbf{0}}_{\mathbf{n/0}} = \dot{\mathbf{0}}^{\mathbf{0}}_{\mathbf{n/0}} + \boldsymbol{\omega(t)} \times \mathbf{p}^{\mathbf{0}}_{\mathbf{n/n}} \tag{48}$$

The expression presented in Equation 48 is useful when expressing the velocities of each link in a robotic manipulator. However it is worth noting that the translational and angular velocity of a given link is dependent on the translational and angular velocities of the previous links. In order to develop the translational velocity contributions for each link the following formula can be utilized [6]:

$$\mathbf{v}^{(\dot{q})}_{cgi/0} = \mathbf{J}^{\mathbf{v_{cgi}}}_{\dot{\mathbf{q}}} \dot{q}$$

$$= \begin{cases} (\boldsymbol{e}^b_p \times \boldsymbol{r}^b_{\mathrm{cgi/p}}) \cdot \dot{q}, & \text{for revolute} \\ \boldsymbol{e}^b_p \cdot \dot{q}, & \text{for prismatic} \end{cases} \tag{49}$$

Here $\mathbf{r_{cgi/p}}$ is the coordinate spanning from the origin of frame p to the center of gravity of the i-th link, while e^b_p is the vector about which the frame p rotates or translates about. Lastly $\dot{q}$ is the time-rate of the generalized coordinate to be inspected. For the case of angular rates one can use the formula:

$$\boldsymbol{\omega}^{(\dot{q})}_i = \mathbf{J}^{\boldsymbol{\omega}_\mathbf{i}}_{\dot{\mathbf{q}}} \dot{q}$$

$$= \begin{cases} \boldsymbol{e}_p^b, & \text{for revolute} \\ 0_{3\times 1}, & \text{for prismatic} \end{cases} \tag{50}$$

4.4 Inverse kinematics and workspace of an anthropomorphic manipulator

Inverse kinematics is the problem of acquiring the joint angles, when the desired end-effector position is known. Using the lengths and angles shown in Figure 15 and given a desired end-effector point $P_d = [Px_d, Py_d, Pz_d]$ the inverse kinematics are given as [10]:

$$\theta_{1d} = atan2(Py_d, Px_d) \tag{51}$$

$$\theta_{2d} = atan2(s2, c2) \tag{52}$$

$$\theta_{3d} = atan2(s3, c3) \tag{53}$$

$$c3 = \frac{Px_d^2 + Py_d^2 + Pz_d^2 - L1^2 - L2^2}{2 \cdot L1 \cdot L2} \tag{54}$$

$$s3 = -\sqrt{1 - c3^2}; \tag{55}$$

$$s2 = \frac{(L1 + L2 \cdot c3) \cdot Pz_d - L2 \cdot s3 \cdot \sqrt{Px_d^2 + Py_d^2}}{Px_d^2 + Py_d^2 + Pz_d^2} \tag{56}$$

$$c2 = \frac{(L1 + L2 \cdot c3) \cdot \sqrt{Px_d^2 + Py_d^2} + L2 \cdot s3 \cdot Pz_d}{Px_d^2 + Py_d^2 + Pz_d^2} \tag{57}$$

It is worth noting that one can only find the desired joint orientations when the desired end-effector point is within the workspace of the robotic manipulator. The workspace can be defined as all possible positions that the manipulator is able to reach. For an anthropomorphic manipulator where one does not account for contact with the ground or other links, the workspace can be represented as the set:

$$\Omega = \left\{ \mathbf{x} \in \mathbb{R}^3 \mid \|\mathbf{x}\| \neq 0 \mid \|L_1 - L_2\| < \|\mathbf{x}\| \leq \|L_1 + L_2\| \right\}. \tag{58}$$

Where $\mathbf{x}$ is the 3×1 vector representing the end-effector position spanning from joint1.

5 Ship and crane integration

This chapter aims to explain how to integrate a ship and crane by the use of quasi-coordinates, as well as discussing the impact a crane will have on the stability of an integrated ship and crane system. Lastly the process of running *20-sim* in parallel with a *.dll* in order to improve simulation time will be explained.

5.1 Integrating a ship and crane by the use of quasi-coordinates

The procedure for creating an integrated crane and vessel model is explained in depth in the paper [6]. This section aims to review the main particulars in order to crate such an integrated model for the Revolt vessel and 3-DOF crane presented in Section 2.1.

In order to create an integrated vessel and crane model, one must first define the generalized coordinates of the system. The generalized coordinates chosen in this thesis can be represented as

$$\mathbf{q} = \begin{bmatrix} \mathbf{r^0_{b/0}} \\ \mathbf{\Theta} \\ \mathbf{q_c} \end{bmatrix}. \tag{59}$$

Here $\mathbf{r^0_{b/0}}$ is the vector representing the surge, sway and heave motions of the ship and $\mathbf{\Theta}$ is the angular displacements along each of the axis of the body fixed frame. Lastly $\mathbf{q_c}$ is the vector containing the angular joint displacements.

The quasi coordinates are defined as time rates of each generalized coordinate defined in terms of the body-fixed frame of the vessel as:

$$\boldsymbol{\omega} = \begin{bmatrix} \mathbf{v^b_{b/0}} \\ \boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/0}} \\ \dot{\mathbf{q}}_{\mathbf{c}} \end{bmatrix}. \tag{60}$$

The terms $\mathbf{v^b_{b/0}}$ and $\boldsymbol{\omega}^{\boldsymbol{b}}_{\boldsymbol{b/0}}$ represent the translational as well as angular velocity vectors of the vessel in the body fixed frame, while $\dot{\mathbf{q}}_{\mathbf{c}}$ represents the vector denoting the angular joint rates of the crane. Recall from Section 3.3 that the quasi coordinates can be found by utilizing a transformation matrix $\boldsymbol{\alpha}$. The complete transformation matrix when including the crane joints becomes:

$$\alpha^T(q) = \begin{bmatrix} \mathbf{R}^b_0 & \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{T}(\Theta)^{-1} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{0}_{3\times3} & \mathbf{I}_{3\times3} \end{bmatrix}. \tag{61}$$

The terms $\mathbf{R}^b_0$ represents the rotation matrix transforming from the inertial to body fixed frame of the vessel, while $\mathbf{T}(\Theta)^{-1}$ is the rotation matrix defined as [6]:

$$\mathbf{T}(\Theta)^{-1} = \left[\mathbf{i}_b, \mathbf{R}^T_x\mathbf{j}_b, \mathbf{R}^T_x\mathbf{R}^T_y\mathbf{k}_b\right]. \tag{62}$$

Where $\mathbf{i}_b$, $\mathbf{j}_b$, $\mathbf{k}_b$ are defined as the unit normal vectors spanning from the body fixed frame:

$$\mathbf{i}_b = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \mathbf{j}_b = \begin{bmatrix} 0 \\ 1 \\ 0 \end{bmatrix}, \mathbf{k}_b = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \tag{63}$$

Additionally, the quasi-coordinate equation of motion presented in Section 3.5 can be rewritten to state space as explained in [6] as:

$$\boldsymbol{\omega} = \mathbf{B}^{-1}\mathbf{p} \tag{64}$$

$$\dot{\mathbf{q}} = \boldsymbol{\beta}\boldsymbol{\omega} \tag{65}$$

$$\dot{\mathbf{p}} = \mathbf{f}_p(\mathbf{q}, \boldsymbol{\omega}) + \boldsymbol{\beta}^{\boldsymbol{T}}\boldsymbol{\tau} \tag{66}$$

The terms $\boldsymbol{\beta}$, $\mathbf{f}_p(\mathbf{q}, \boldsymbol{\omega})$ and $\mathbf{B}$ are defined as:

$$\boldsymbol{\beta} = (\boldsymbol{\alpha}^T)^{-1} \tag{67}$$

$$\mathbf{f}_p(\mathbf{q}, \boldsymbol{\omega}) = \boldsymbol{\beta}^T\boldsymbol{\gamma}\mathbf{B}\boldsymbol{\omega} + \frac{1}{2}\boldsymbol{\beta}^T\boldsymbol{\omega}^T\frac{\partial \mathbf{B}}{\partial \mathbf{q}}\boldsymbol{\omega} + \mathbf{C}_\mathbf{A}(\boldsymbol{\omega})\boldsymbol{\omega} \tag{68}$$

$$\mathbf{B} = \mathbf{B}_b + \sum_{i=1}^{3}\mathbf{B}_i \tag{69}$$

Where $\mathbf{C}_\mathbf{A}(\boldsymbol{\omega})$ is the hydrodynamic coriolies and centripetal matrix, $\mathbf{B}_b$ is the mass matrix of the ship expressed in terms of the body-fixed frame of the ship, while $\mathbf{B}_i$ are the mass matrices of each crane links. These are defined as:

$$\mathbf{B}_b = \mathbf{J}_b^T \begin{bmatrix} \mathbf{M} & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I_g} \end{bmatrix} \mathbf{J}_b. \tag{70}$$

$$\mathbf{B}_i = \mathbf{J}_i^T \begin{bmatrix} \mathbf{M}_i & \mathbf{0}_{3\times3} \\ \mathbf{0}_{3\times3} & \mathbf{I}_i^b \end{bmatrix} \mathbf{J}_i^T. \tag{71}$$

Here $\mathbf{I}_g$ is the inertia tensor of the ship relative to the center of gravity, while $\mathbf{I}_i^b$ is the inertia tensor of the i-th link expressed in terms of the body fixed frame of the vessel.

The matrix $\mathbf{M}$ is defined as

$$\mathbf{M} = m\mathbf{I}_{3\times3} \tag{72}$$

where m is the mass of the ship. Additionally $\mathbf{M}_i$ is defined in a similar manner as

$$\mathbf{M_i} = m_i\mathbf{I}_{3\times3} \tag{73}$$

where m_i is the mass of the i-th link. Lastly the geometric jacobians $\mathbf{J}_b$ and $\mathbf{J}_i$ are defined as:

$$\mathbf{J_b} = \begin{bmatrix} \mathbf{J_b^v} \\ \mathbf{J_b^\omega} \end{bmatrix} \tag{74}$$

$$\mathbf{J_b^v} = \begin{bmatrix} \mathbf{I_{3\times 3}} & \mathbf{i_b} \times \mathbf{r^b_{cg/0}} & \mathbf{j_b} \times \mathbf{r^b_{cg/0}}, & \mathbf{k_b} \times \mathbf{r^b_{cg/0}} & \mathbf{0}_{3\times 3} \end{bmatrix} \tag{75}$$

$$\mathbf{J_b^\omega} = \begin{bmatrix} \mathbf{0}_{3\times 3} & \mathbf{I}_{3\times 3} & \mathbf{0}_{3\times 3} \end{bmatrix} \tag{76}$$

$$\mathbf{J_i} = \begin{bmatrix} \mathbf{J_i^v(q)} \\ \mathbf{J_i^\omega(q)} \end{bmatrix}. \tag{77}$$

Here $\mathbf{J}_b$ is the geometric jacobian matrix that allows one to express the kinetic enegry of the ship in terms of quasi coordinates, while $\mathbf{J}_i$ is the geometric jacobian of the i-th link expressed in terms of the body-fixed frame of the vessel.

5.2 Simulation setup by use of external *.dll*

In order to connect a ship and crane there are two main approaches. One can either create an integrated model by utilizing potential and kinetic energy and the Lagrangian equations of motion, or connect the two models by the use of very stiff springs. One of the reasons creating an integrated model is more preferred than simply utilizing stiff springs is due the increase of computational power. However even an integrated model can also be computationally expensive to simulate.

In order to combat this, one can utilize the math software *Maple* in order to find the kinetic energy of the system, and export key parameters such as the partial derivatives of the mass and alpha matrix presented in Section 5 to C-code. One can then utilize the integrated development environment (IDE) *Visual Studio* in order to compile a *.dll*, which is an executable that can run in parallel with a the simulation software *20-sim*.

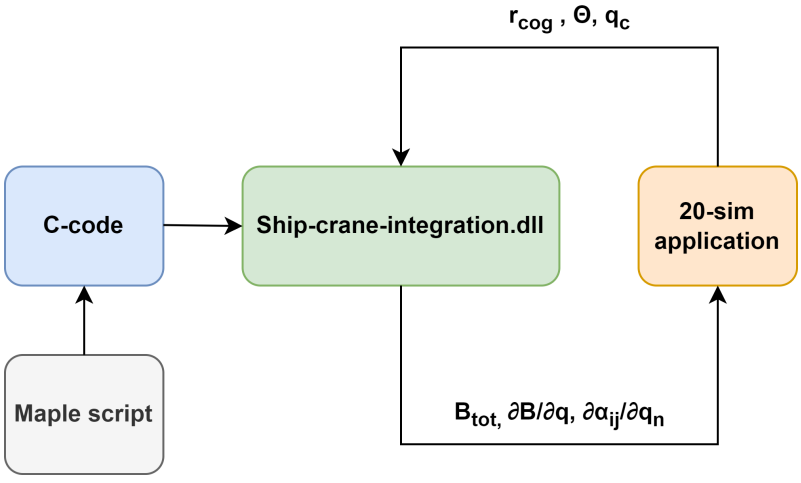


Figure 16: Block diagram of *.dll* setup in parallel with 20-sim used in thesis

In Figure 16 the parameters $\mathbf{r_{cog}}$ and $\mathbf{\Theta}$ are the position of the center of gravity in terms of the body-fixed frame, and the angular displacement vector of the ship. Additionally, $\mathbf{q_c}$ is the vector containing the crane joint displacements. The arguments $\mathbf{B_{tot}}$, $\partial\mathbf{B}/\partial\mathbf{q}$ and $\partial\alpha_{ij}/\partial q_n$ are the total mass matrix of the system, the partial derivatives of the mass matrix with regards to generalized coordinates and the element-wise partial derivatives of the alpha matrix with respect of to the generalized coordinates.

5.3 Stability properties of a ship and crane system

The following section is based on [12], where the stability properties of a ship is thoroughly discussed. Recall from Section 2.1 where the restoring moments in roll and sway was briefly touched upon. The moment arm GZ is a function of the GM value as shown in the formula

$$GZ = sin(\theta)GM. \tag{78}$$

The GM value is of particular note when inspecting the stability properties of a ship, and can be defined as

$$GM = KM - VCG \tag{79}$$

where KM is the distance from the keel to the metacentre, and VCG is the vertical position of the centre of gravity. As discussed in [12], in order for a ship to be stable the GM value needs to be larger than zero. Intuitively this makes sense when inspecting Equation 78, as a GM value equal to zero will give zero restoring forces, while negative GM values will induce moments that will lead a vessel to capsize. Additionally one can see that the larger the GM value is, the larger the restoring moment will be.

By inspecting Equation 79 it becomes clear that the GM value is dependent on the vertical center of gravity. The higher the center of gravity is relative to the keel, the smaller the GM-value becomes. This makes the stability properties of a ship different before and after mounting a shipboard crane.

Attaching a crane on the deck of the ship will move the vertical center of gravity upwards. Additionally, if the crane is able to move, then the vertical center of mass will vary based on the crane configurations. Should the crane also be carrying a load, then the vertical center of gravity will also be further impacted, thus further reducing the restoring moments. Depending on the initial stability of the ship, this can result in large roll angles.

Another thing to note is how GZ-curves are generated. A single GZ-curve will show the restoring arm for a specific vertical center of mass and pitch angle, and in order to get accurate GZ-values for a system consisting of a shipboard crane, one would need to generate a separate GZ curve for each change in the center of gravity and pitch angle. A simplification to avoid this is to choose a single GZ-curve when simulating a system where the roll and pitch angles have larger responses than 10°.

6 Control

In this chapter the PID-controller as well as the different control schemes used in the thesis will be explained. The main algorithms to be described are: Point-to-point motion, sway compensation and crane-tip station-keeping and lastly wire-lowering and heave compensation.

6.1 PID controllers

A PID-controller, short for proportional, integral and derivative controller is commonly used in the thesis. The resulting control law can take two forms. The two different forms are

$$u = K_p e + K_d \dot{e} + K_i \int e dt \tag{80}$$

and

$$u = K_p e + K_p/T_d \dot{e} + K_p/T_i \int e dt. \tag{81}$$

It can be seen that there are two ways of writing the integral and derivative coefficients, as $K_i = K_p/T_i$ and $K_d = K_p/T_d$, where T_i and T_d is the integral and derivative time respectively [19]. The PID-controller has three different gains. The proportional gain $K_p \cdot e$ allows the controller to quickly converge towards the desired reference. The integral gain $K_p/T_i \cdot \int e$ integrates the error over time, and can eliminate offsets that can occur with only proportional gain in steady state. The derivative gain $K_p/T_d \cdot \dot{e}$ will give a larger contribution to the output for rapid changes in the error, effectively damping the system. In Figure 17 the block diagram of a PID-controller is presented.

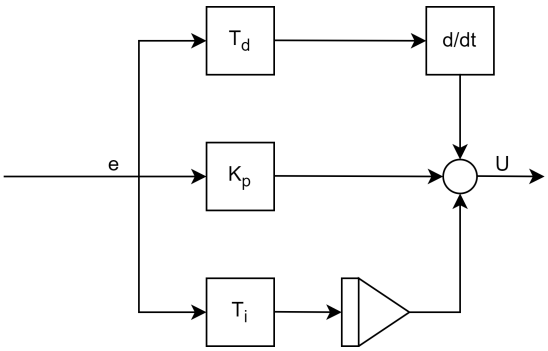


Figure 17: Block diagram of PID-controller. Also discussed in [19].

6.2 Point-to-point motion of a knuckle-boom crane

In order to perform point-to-point motion with the 3 DOF crane, one must first find the desired joint angles of the crane in order to reach a given position in the world. This can be done by developing expressions for the inverse kinematics for the crane. Once the inverse kinematics are available, one needs to generate a velocity profile describing the motion from the initial joint configuration of the crane, to the end configuration found from the inverse kinematics. A block diagram for performing point to point motion is shown in Figure 18.

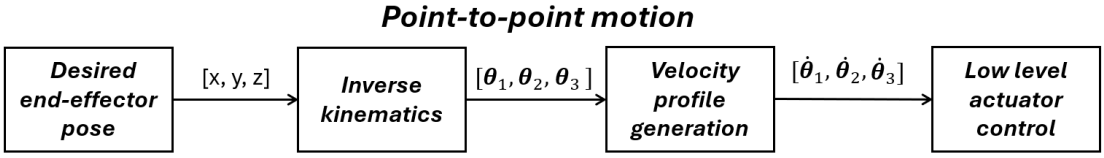


Figure 18: Block diagram of point-to-point motion [10].

According to [10] one can choose a third order polynomial in order to determine the joint motion:

$$q(t) = a_3 t^3 + a_2 t^2 + a_1 t + a_0. \tag{82}$$

In order to determine the coefficients a_3, a_2, a_1 and a_0 the polynomial need to satisfy four constraints. These are the initial and end angles q_i, q_f, as well as the initial and end velocities $\dot{q}_i$, $\dot{q}_f$. Additionally the time for the manipulator to perform the motion t_f also needs to be specified. The following constraint equations to be solved then becomes:

$$\begin{aligned} a_0 &= q_i && (83)\\ a_1 &= \dot{q}_i && (84)\\ a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 &= q_f && (85)\\ 3a_3 t_f^2 + 2a_2 t_f + a_1 &= \dot{q}_f && (86)\\ & && (87) \end{aligned}$$

However, in a loading operation controlling the start and end accelerations of the crane tip can be of use. This is because smaller accelerations can allow for smaller pendulum motions of the cargo attached to the wire. In order to choose the desired start and end accelerations, a fifth order polynomial needs to be chosen. This gives six constraint equations to be solved. The polynomial that represents the joint motion then becomes [10]:

$$q(t) = a_5 t^5 + a_4 t^4 + a_3 t^3 + a_2 t^2 + a_1 t + a_0. \tag{88}$$

Meanwhile the constraint equations consisting of initial as well as end positions and velocities, now also contain initial and end acceleration as well. The constraint equations to be solved then becomes [10]:

$$
\begin{aligned}
a_0 &= q_i \\
a_1 &= \dot{q}_i \\
2a_2 &= \ddot{q}_i && (89)\\
a_5 t_f^5 + a_4 t_f^4 + a_3 t_f^3 + a_2 t_f^2 + a_1 t_f + a_0 &= q_f && (90)\\
5a_5 t_f^4 + 4a_4 t_f^3 + 3a_3 t_f^2 + 2a_2 t_f + a_1 &= \dot{q}_f && (91)\\
20a_5 t_f^3 + 12a_4 t_f^2 + 6a_3 t_f + 2a2 &= \ddot{q}_f && (92)\\
& && (93)
\end{aligned}
$$

Solving the constraint equations for the fifth order polynomial then allows one to generate the velocity profile to each joint of the manipulator as [10]

$$\dot{q}_i = 5a_5 t^4 + 4a_4 t^3 + 3a_3 t^2 + 2a_2 t + a_1 \tag{94}$$

where the coefficients a_5, a_4, a_3, a_2, a_1 and a_0 needs to be solved for each joint velocity profile $\dot{q}_i$. Utilizing Equation 94, one can the perform point-to-point motion with the manipulator. However the the trajectory q_i only satisfies the aforementioned constraints, and is thus not necessarily the most optimal path the manipulator can take in order to reach the end-configuration.

6.3 Sway compensation and crane-tip station-keeping

The problem of sway compensation is to dissipate the kinetic energy of the load by steering the movements of the crane tip and adjusting the wire length [20]. In this thesis the control problem of sway compensation of a crane-tip can be categorized into three main components. The first problem is station-keeping, as the desired crane tip location is not to diverge from the wanted end-effector position. The second control problem is to minimize the pendulum motions of the spreader mechanism which is attached to the crane tip by wire. This can be done by minimizing the moment arm of the pendulum by controlling the crane tip to be in the same x and y position of the spreader. Lastly one needs to translate translational motion in x, y and z to desired joint angles.

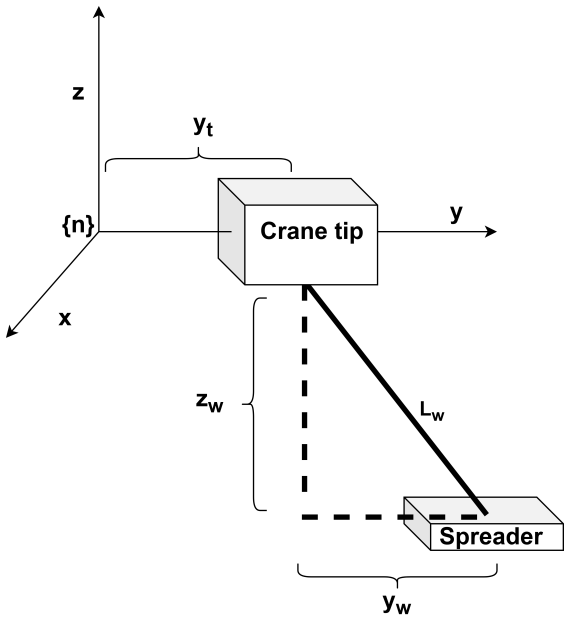


Figure 19: Principle sketch for sway compensation in y-direction

In Figure 19 the moment arm of the crane-tip and spreader system is shown as y_w, while the crane tip position relative to a wanted end effector position is shown as y_t. In order to make y_w, and y_t go towards zero as time goes to infinity PID-controllers are used. The control law for sway compensation in y-direction then becomes:

$$u_{sway-y} = K_p e_y + K_d \dot{e}_y + K_i \int e_y dt = K_p[y_{spreader} - y_{tip}] + K_d[y_{spreader} - y_{tip}] \frac{d}{dt} + K_i \int [y_{spreader} - y_{tip}] dt. \tag{95}$$

In order to expand the control laws to include x - direction, one needs to include another DOF. This can be simply done by using the same errors, but inspecting the x positions instead.

$$u_{sway-x} = K_p e_x + K_d \dot{e}_x + K_i \int e dt = K_p[x_{spreader} - x_{tip}] + K_d[x_{spreader} - x_{tip}] \frac{d}{dt} + K_i \int [x_{spreader} - x_{tip}] dt. \tag{96}$$

Additionally the control law for station-keeping of the crane tip in x, y and z direction becomes:

$$u_{stationkeeping-x} = K_p e_x + K_d \dot{e}_x + K_i \int e_x dt = K_p[x_d - x_m] + K_d[x_d - x_m] \frac{d}{dt} + K_i \int [x_d - x_m] dt. \tag{97}$$

$$u_{stationkeeping-y} = K_p e_y + K_d \dot{e}_y + K_i \int e_y dt = K_p[y_d - y_m] + K_d[y_d - y_m] \frac{d}{dt} + K_i \int [y_d - y_m] dt. \tag{98}$$

$$u_{stationkeeping-z} = K_p e_z + K_d \dot{e}_z + K_i \int e_z dt = K_p[z_d - z_m] + K_d[z_d - z_m] \frac{d}{dt} + K_i \int [z_d - z_m] dt. \tag{99}$$

Here x_m, y_m and z_m represents the measured crane tip position and x_d, y_d and z_d are the desired crane tip positions. K_p, K_d and K_i the proportional, derivative and integral controller gains for

each DOF, which do not need to be identical.

In order to accomplish both station-keeping and moment minimization, a control system that utilized both station-keeping and moment-minimization in parallel was chosen. With the controllers having the same proportional and derivative gains in order to not out-compete each-other. The resulting control input for the crane then becomes:

$$\mathbf{u}_{tot} = \begin{bmatrix} u_{sway-x} + u_{stationkeeping-x} \\ u_{sway-y} + u_{stationkeeping-y} \\ u_{stationkeeping-z} \end{bmatrix} \tag{100}$$

Lastly, in order to generate translational motion of the crane tip, the geometric jacobian relating the joint velocities to the translational end-effector velocities was chosen. By inverting the geometric jacobian one could find the desired joint angles that would induce the wanted translational motion. The resulting control algorithm can then be seen in Figure 20.

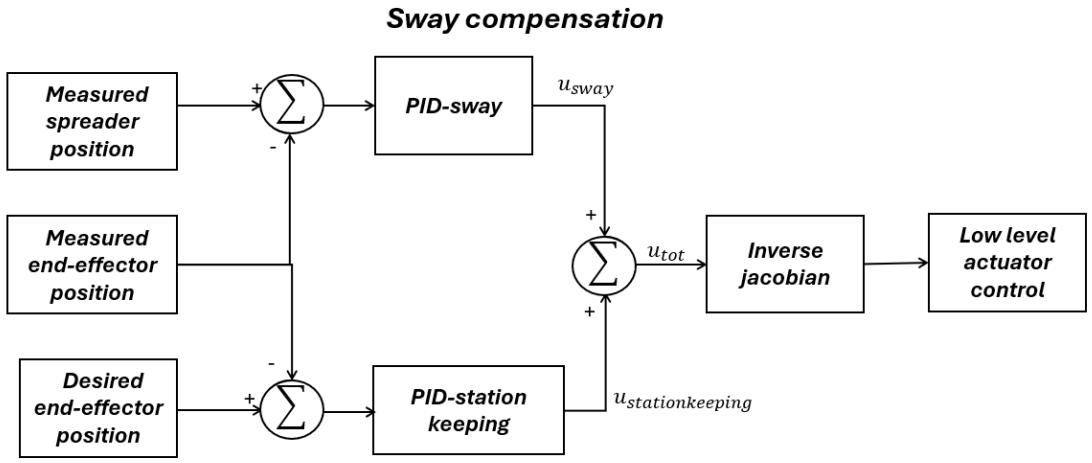


Figure 20: Block diagram of sway compensation and crane-tip station-keeping

6.4 Wire lowering and heave compensation

Vessels at sea encounters environmental forces in the form of wind, waves and current. Out of these waves in particular can induce unwanted vertical motions on cargo being transported by the use of a shipboard crane. The minimization of the vertical displacements induced by heave motions is called heave compensation and can take the form of both passive as well as active heave compensation. Passive heave compensation involves utilizing spring damper systems which act as load absorbers, which in turn attenuates heave motion of a given load. Meanwhile active heave compensation involves a control system compensating for heave motion by the use of actuators [21].

In this thesis a control scheme for active heave compensating by utilizing wire control has been considered. In order to minimize the heave induced motions that the ship has on the cargo attached to the crane tip the desired reference of the winch was set to be the opposite motion that would be generated in heave, as this would cancel out the relative motion of the spreader mechanism. Additionally the problem of lowering the cargo while the vessel experienced heave motions was also

considered. In order to lower the spreader and attached cargo to the ground, while compensating for heave motions, the winch controller was given a reference composed of both the opposing heave motion reference added together with a reference generated from a reference filter. The final reference is then fed into a PID controller.

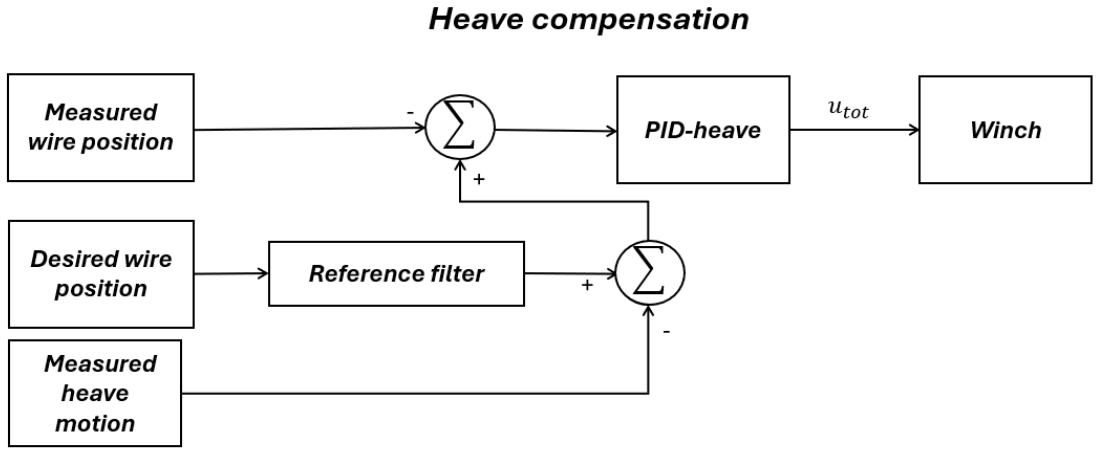


Figure 21: Block diagram of heave and wire control

A reference filter allows for references that a controller can more easily follow by generating a curve from the initial system position to a desired end position. The reference filter chosen for wire control is taken from [22] where it is intended to generate references for a ship performing dynamic positioning in the earth fixed reference frame:

$$\begin{aligned} \mathbf{a}_d^e + \mathbf{\Omega}\mathbf{v}_d^e + \mathbf{\Gamma}\mathbf{x}_d^e &= \mathbf{\Gamma}\mathbf{x}_{ref} \\ \dot{\mathbf{x}}_{ref} &= -\mathbf{A}_f\mathbf{x}_{ref} + \mathbf{A}_f\boldsymbol{\eta}_r \end{aligned} \tag{101}$$

The terms $\mathbf{x}_{ref}$ $\dot{\mathbf{x}}_{ref}$ is the reference and its timerate. Additionally, $\mathbf{a}_d^e$, $\mathbf{v}_d^e$ and $\mathbf{x}_d^e$ represent the desired acceleration, velocities and position trajectories in the earth fixed frame, while $\boldsymbol{\eta}_r$ is the vector denoting new reference coordinates. The matrices $\mathbf{\Omega}$, $\mathbf{\Gamma}$ and $\mathbf{A}_f$ is the non-negative damping matrix, the diagonal stiffness matrix and a first order diagonal and non-negative set-point filter gain matrices respectively. Each matrix are defined as [22]

$$\mathbf{\Omega} = diag\{2\zeta_i\omega_i\} \tag{102}$$

$$\mathbf{\Gamma} = diag\{\omega_i\} \tag{103}$$

$$\mathbf{\Omega} = diag\{1/t_i\} \tag{104}$$

were ζ_i, ω_i and t_i are tuning parameters.

However, as the wire control algorithm has 1 DOF, the equations presented in Equation 101 becomes scalars. Additionally the reference frame used in the winch is simply the inertial reference frame, this simplifies the earlier equations to be:

$$a_d^0 + \Omega v_d^0 + \Gamma x_d^0 = \Gamma x_{ref}$$
$$\dot{x}_{ref} = -A_f x_{ref} + A_f \eta_r. \tag{105}$$

6.5 Integrated control system

The integrated control system was created with the intent of being able to transport cargo from an initial position, to a given end position. The cargo in question can either be the spreader itself, or a spreader connected to a container. In order to accomplish this the integrated system makes use of an FSM in order to toggle between the different states, to which different behaviors are associated.

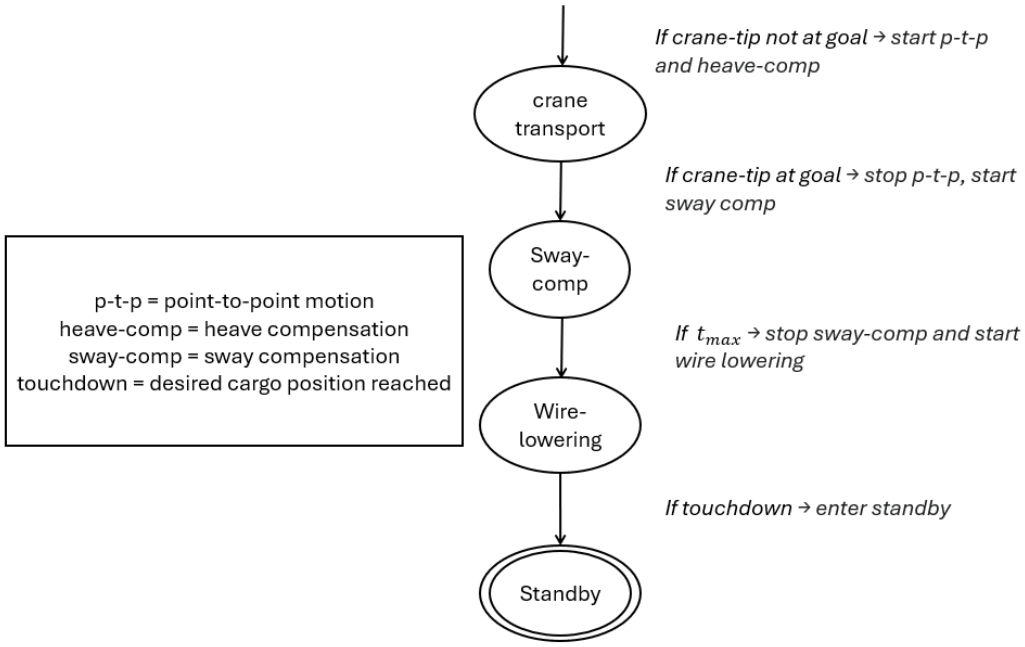


Figure 22: State diagram of FSM

The state diagram shown in Figure 22 shows how each state is toggled between, as well as the transitions between each state. Each state is denoted as a circle, and the final state is shown as a double circle.

In order to evaluate if the crane-tip has reached its desired end-goal the combined position error in each DOF is used:

$$e_x = |x_d - x_m|$$
$$e_y = |y_d - y_m|$$
$$e_z = |z_d - z_m|$$
$$CombinedError = e_x + e_y + e_z \tag{106}$$

Here x_m, y_m, z_m are the measured x, y and z positions, while x_d, y_d, z_d are the desired positions

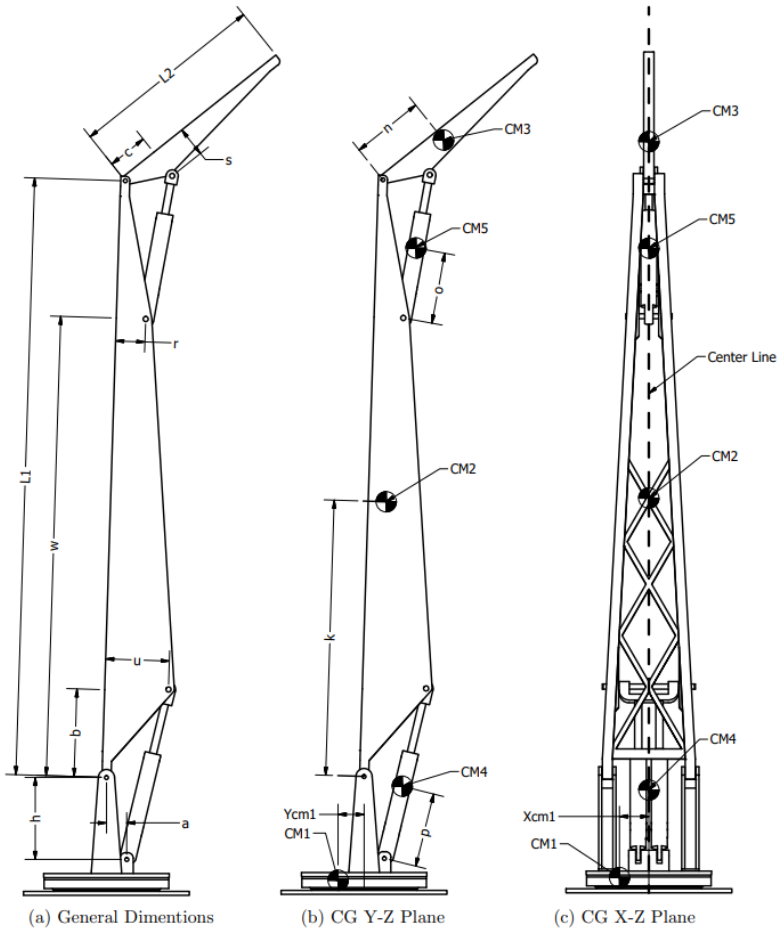
in each DOF. The combined error is then evaluated against an acceptable error margin which can be adjusted. Additionally, the sway-compensation and station-keeping algorithm is toggled by the use of a timer. If the simulation-time since the sway-compensation algorithm initiated surpasses a given t_{max}, then the algorithm will stop and enter the next state. Lastly the desired cargo position is determined to be reached if the combined error described in Equation 106 is sufficiently small.

7 Modeling

The following chapter aims to give an overview of the implementation of the physical structures as well as the control algorithms presentment in the thesis. Closer details on the implemented code, and control algorithms are presented in the appendix.

7.1 Assumptions, simplifcations and model parameters

The geometric crane parameters for the downscaled crane modeled by Gyberg [4] is presented in Table 2 and Figure 23. Due to the small size of the crane, it was chosen to scale it by a factor of 6,5. This increased the length of the crane links to 18.85 $[m]$ for the lower arm and 6,5 $[m]$ for the upper arm, as seen in Table 3, providing a reach of around 20 $[m]$. This was also done in order to improve the carrying capacity from 16 $[kg]$ to several tonnes, as it was found that in that cranes with reach spanning around 20 $[m]$ is able to carry 9-6 $[tonn]$ [23]. However the exact weight the upscaled crane model would be able to carry is unknown, as this is dependent on the force of the crane actuators, and material strength of the crane.



(a) General Dimentions (b) CG Y-Z Plane (c) CG X-Z Plane

Figure 23: All crane lengths [4].

Downscaled crane parameters	Size	Unit
$Capacity$	16	$[kg]$
$L1$	2900	$[mm]$
$L2$	1000	$[mm]$
h	1450	$[mm]$
a	100	$[mm]$
b	500	$[mm]$
u	300	$[mm]$
r	150	$[mm]$
w	2000	$[mm]$
c	200	$[mm]$
s	150	$[mm]$
$Ycm1$	0	$[mm]$
$Xcm1$	0	$[mm]$
k	500	$[mm]$
n	400	$[mm]$
p	400	$[mm]$
o	400	$[mm]$

Table 2: Key crane parameters of downscale knuckle-boom crane [4].

Upscaled crane parameters	Size	Unit
$L1$	18850	$[mm]$
$L2$	6500	$[mm]$
h	9425	$[mm]$
a	650	$[mm]$
b	3250	$[mm]$
u	1950	$[mm]$
r	975	$[mm]$
w	13000	$[mm]$
c	1300	$[mm]$
s	975	$[mm]$
$Ycm1$	0	$[mm]$
$Xcm1$	0	$[mm]$
k	3250	$[mm]$
n	2600	$[mm]$
p	2600	$[mm]$
o	2600	$[mm]$

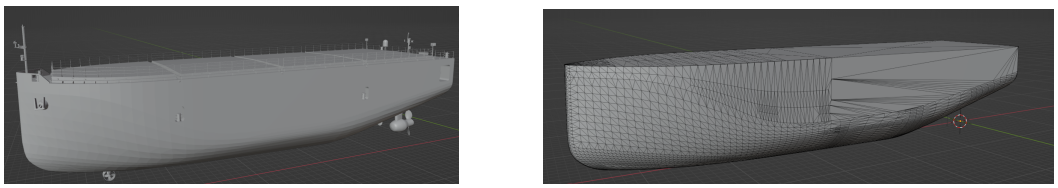
Table 3: Geometric crane parameters used in simulation. Scaled by 6,5.

The crane actuators was also simplified to become Sf elements providing joint velocities, and the mass and inertia of the piston actuators were removed from the model. This choice was done firstly due to the thesis focus being high level control and modeling of an integrated simulation model capable of performing a loading operation. Secondly the *Maple* script used in the ship and crane integration was not able to take the partial derivatives of the mass matrices when the terms from the crane actuators were included. The resulting crane actuators therefore lack any delay, respond instantly and is able to carry any weight the crane or spreader should have.

Furthermore, the spreader mechanism was simplified to be a box with the dimensions width = 2.44 $[m]$, height = 0.30 $[m]$ and length = 6.10 $[m]$. The inertia was then found by assuming it to be made of a homogeneous material, where the total mass was assumed to be 1.5 $[tonn]$. The wire was simplified to be an overdamped 3D-spring, and controlled by a simplified winch which is

modeled as a MSf element.

Additionally the GM-value of in roll was found to be larger than 0 even in the worst loading configuration as shown in Section D. The GM-value of the ship in roll direction was found be reduced from GM = 0.387 $[m]$ to GM = 0.160 $[m]$. In order to find an expression of the restoring coefficient in roll C_{44}, the mean of these GM-values was used, which gave a resulting restoring coefficients to be: $GM = 0.273$ and $C_{33} = 6.442 \cdot 10^6$.



(a) Hull received from DNV.

(b) Simplified Revolt hull.

Figure 24: Revolt hull before and after pruning.

All the coefficients and rotational inertias of the ship is derived from a 3D model of the Revolt vessel provided by DNV, which can be seen in Figure 24a. Since the model provided by DNV could not be imported into the computer aided design (CAD) software *Autocad Inventor* due to its large size, work was done to simplify the model. After pruning the 3D model such that only the simplified hull seen in Figure 24b was left, it was fed into both *Autocad Inventor* as well as the ship stability computation program $Delftship$ in order to find reasonable estimates of the inertias as well as hydrostatic coefficients associated with the vessel.

The inertia tensor of Revolt was found by assuming the hull to be a homogeneous material. Then the material density that corresponded with the ship weight and volume occupied by the hull was used, as detailed information of the hull thickness was unavailable. In order to include added mass, curves showing added mass at different excitation frequencies for a conventional ship in [17] was used. The added mass coefficients was chosen at $\omega \approx 0$, as this corresponded with the wave frequencies found in the case. Lastly hydrodynamic damping was tuned by inspecting the what a reasonable time constant of the revolt ship would be.

7.1.1 Main system parameters

Properties	Revolt	Units
$Mass$	2400	$[tonn]$
I_{xx}	5.902 $\cdot 10^7$	$[kgm^2]$
I_{yy}	5.870 $\cdot 10^8$	$[kgm^2]$
I_{zz}	5.986 $\cdot 10^8$	$[kgm^2]$
I_{yx}	1852	$[kgm^2]$
I_{zx}	9.621 $\cdot 10^6$	$[kgm^2]$
I_{zy}	-527	$[kgm^2]$
Loa	60	$[m]$
B	14.5	$[m]$
H	12.5	$[m]$
$Draught$	5.045	$[m]$
C_{33}	$4.667 \cdot 10^6$	$[Nm/rad]$
C_{44}	$6.442 \cdot 10^6$	$[Nm/rad]$
C_{55}	$1.624 \cdot 10^9$	$[Nm/rad]$
R_{11}	$150 \cdot 10^6$	$[Ns/m]$
R_{22}	$28.6 \cdot 10^6$	$[Ns/m]$
R_{33}	$1 \cdot 10^9$	$[Ns/m]$
R_{44}	$38.4 \cdot 10^6$	$[Nms/rad]$
R_{55}	$1 \cdot 10^{10}$	$[Nms/rad]$
R_{66}	$16.14 \cdot 10^6$	$[Nms/rad]$
A_{11}	$1.4 \cdot 10^6$	$[kg]$
A_{22}	$7.2 \cdot 10^6$	$[kg]$
A_{33}	$11.1 \cdot 10^7$	$[kg]$
A_{44}	$1.75 \cdot 10^8$	$[kgm^2]$
A_{55}	$8 \cdot 10^8$	$[kgm^2]$
A_{66}	$4.2 \cdot 10^9$	$[kgm^2]$

Table 4: Revolt parameters used in simulation.

Properties	Base	Lower arm(Link1)	Upper arm (Link2)	Units
$Mass$	27515,861	13648,269	1416,126	$[kg]$
I_{xx}	60595,194	366338,681	5984,350	$[kgm^2]$
I_{yy}	64717,665	16050,499	173,304	$[kgm^2]$
I_{zz}	84633,157	372589,19	5851,933	$[kgm^2]$

Table 5: Mass and rotational inertia for the knuckle-boom crane.

Properties	Spreader	Units
$Mass$	1.5 $\cdot 10^5$	$[kg]$
I_{xx}	1.3272 $\cdot 10^8$	$[kgm^2]$
I_{yy}	7.3782 $\cdot 10^8$	$[kgm^2]$
I_{zz}	6.880 $\cdot 10^8$	$[kgm^2]$
I_{xz}	7.200 $\cdot 10^6$	$[kgm^2]$

Table 6: Mass and rotational inertia for the spreader mechanism [5].

7.2 All coordinate frames of integrated system

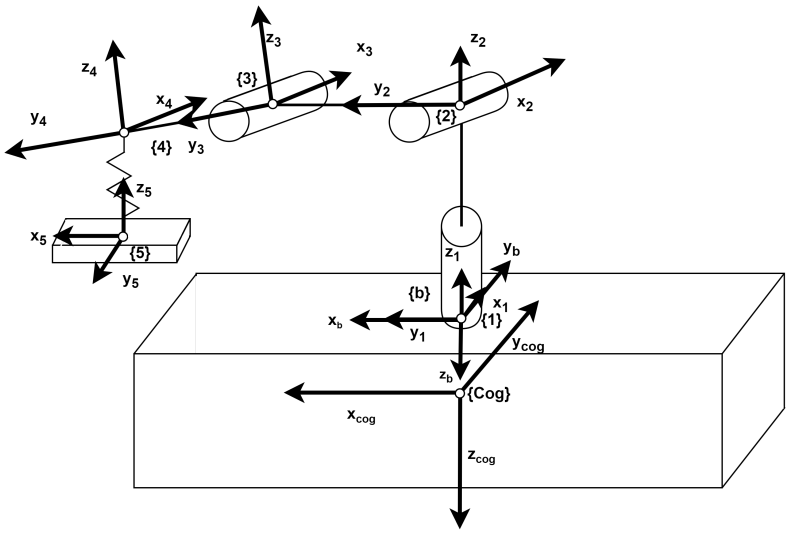


Figure 25: All coordinate frames of complete system

In Figure 25 Revolt has been simplified to be a box, while the 3-DOF knuckle-boom crane is simplified to be an anthropomorphic manipulator. As seen in Figure 25, the final system consists of 7 different coordinate systems. The frames related to the vessel is **{Cog}** and **{b}**, which represent the frame associated with the center of gravity and the body-fixed frame of the ship. The body-fixed frame is located in the middle of the deck of Revolt, and it is important to note that the center of gravity does not coincide with the body-fixed frame. This is was taken into account in Section 5, when the mass matrix of the ship was defined, and the center of mass is also included as part of the arguments the *.dll* receives when computing the mass matrix and its partial derivatives.

The crane joint frames remain mostly the same as in [4], but the frames associated with the piston actuators have been removed as the crane model has been simplified. The frames **{1}**, **{2}** , **{3}** and **{4}** correspond to the base frame, lower joint frame, upper joint frame and the crane tip frame respectively. Lastly, the spreader and its frame **{5}** is also shown. Since slew control by the use of tugger wires is outside the scope of the thesis, the rotations of the spreader frame coincides with the inertial frame.

7.3 Bond graph model of complete system

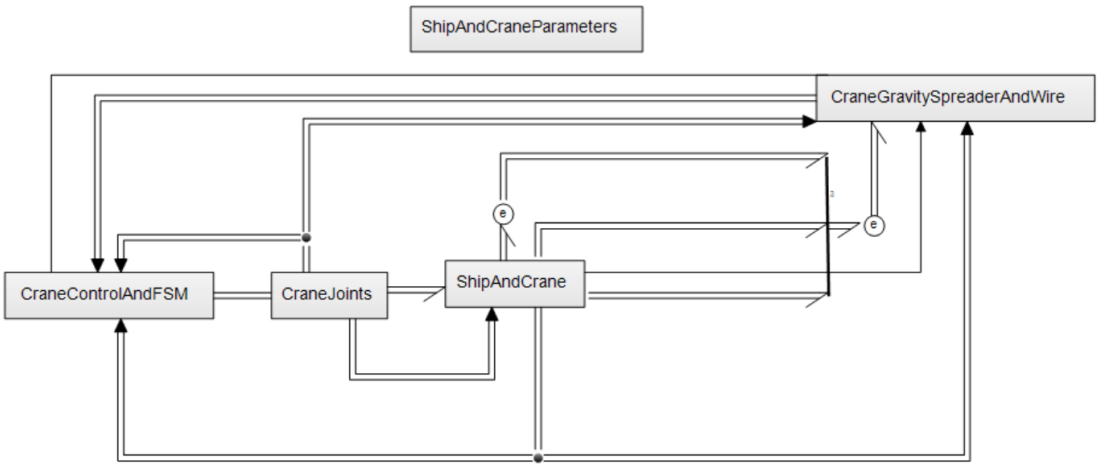


Figure 26: Complete system

Figure 26 shows the complete system, and due to the system complexity it has been split into several sub-models. The ***ShipAndCrane*** sub-model contains the integrated bond graph model of the crane and ship. The ***CraneJoints*** block finds the joint positions based on the the desired joint velocities delivered from the control system. The ***CraneControlAndFSM*** block contains the finite state machine which is responsible for toggling between the different behaviors of the system, as well as the control algorithms enabling point-to-point motion, crane-tip station-keeping and sway compensation.

Additionally a power-muxer can be seen. This takes in 3 3-dimensional power bonds, and combines it into a single 9-dimensional power bond representing the quasi-coordinates of the system. This power bond is then fed into the block named ***CraneGravitySpreaderAndWire*** which contains the gravitational forces acting on the crane tip and each of the links. The submodel then converts the gravitational forces into forces and moments which act on the vessel in the body-fixed frame. Lastly, the submodel also contains the winch controller as well as the spreader and wire model.

7.4 Ship and crane model

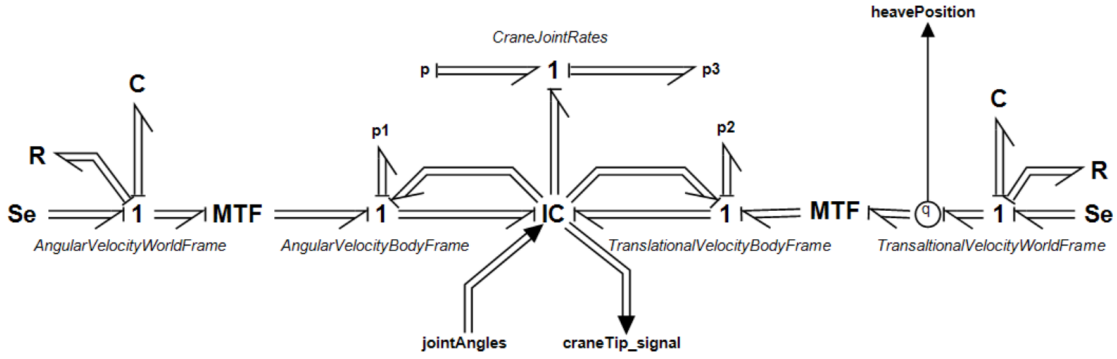


Figure 27: Bond graph model of ship and crane

Figure 27 shows the implementation of the **IC**-field which is responsible for calculating the quasi-coordinate equations of motion of the integrated system. The **IC**-field runs a *.dll* file in parallel to the *20-sim* application in order to compute the state space equations of the 9 DOF ship-crane model.

The **MTF** elements in Figure 27 transforms from the world to the body-fixed reference frame of the vessel, and it can be seen that the hydrodynamic restoring forces has been implemented as linear **C**-elements, while hydrodynamic damping is implemented as linear **R**-elements.

The forces and moments that act on the integrated ship-crane system due to the weight of each of the crane-links as well as the attached cargo interface through the power ports **p1** and **p2** respectively, and have been transformed to the body-fixed frame of the vessel beforehand in the block ***CraneGravitySpreaderAndWire***.

Lastly there are several signals both received and sent out of the model. The ***craneTip_signal*** sends out the updated crane-tip position to both the control system and the wire and spreader model. The ***heavePosition*** signal contains the heave motions of the ship, and is sent into the winch controller for heave compensation. Lastly the ***jointAngles*** signal contain the joint positions for all the crane joints. This is used in the **IC**-field in order to compute the newest crane position and is also provided to the *.dll* in order to solve the system equations.

7.5 Gravity

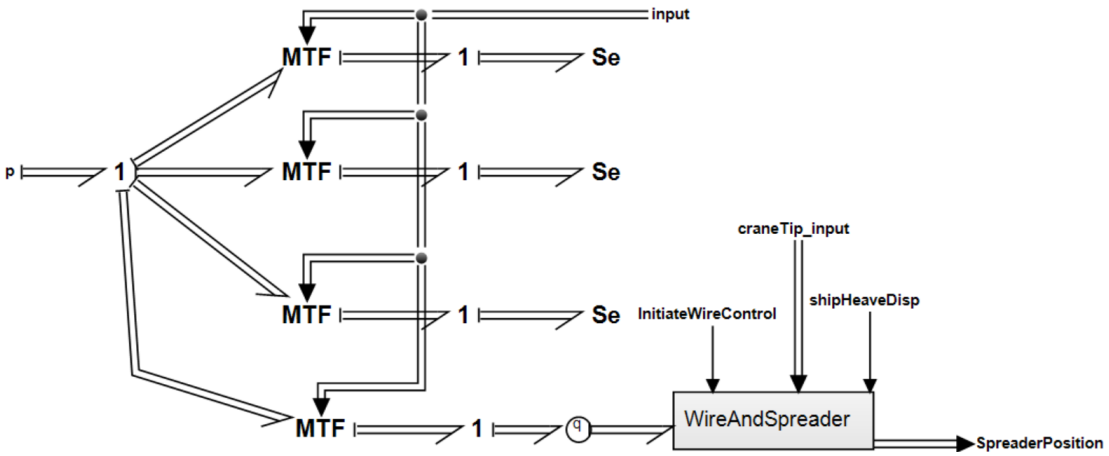


Figure 28: Bond graph model of gravity and spreader model

Figure 28 shows how the gravitational force for each of the crane links as well as the weight of the spreader mechanism is transformed to forces and moment to the body-fixed frame of the vessel. Each of the **MTF** elements utilize the geometric jacobian relating the quasi coordinates to the center of mass for each link, while also transforming between the world and body-fixed frame of the vessel. The **Se** elements shown represent gravitational forces of the base and upper and lower crane links respectively. The gravitational forces are related to the body fixed frame in the following manner:

$$\boldsymbol{\tau_{cm1}} = \mathbf{J_{cm1}}^T \mathbf{R_0^b F_{cm1}} \tag{107}$$

$$\boldsymbol{\tau_{cm2}} = \mathbf{J_{cm2}}^T \mathbf{R_0^b} \mathbf{F_{cm2}} \tag{108}$$

$$\boldsymbol{\tau_{cm3}} = \mathbf{J_{cm3}}^T \mathbf{R_0^b} \mathbf{F_{cm3}} \tag{109}$$

$$\boldsymbol{\tau_{tip}} = \mathbf{J_{tip}}^T \mathbf{R_0^b} \mathbf{F_{tip}} \tag{110}$$

The force vectors $\boldsymbol{\tau}_{cm1}$, $\boldsymbol{\tau}_{cm2}$, $\boldsymbol{\tau}_{cm3}$, and $\boldsymbol{\tau}_{tip}$ are the 9×1 force vectors corresponding to a force in each of the DOFs of the integrated ship-crane system as a result of the gravitational forces. These attack each crane joint as well as all 6 DOFs of the ship after being decomposed by the power muxer seen in the integrated model.

The terms $\mathbf{J}_{cm1}$, $\mathbf{J}_{cm2}$, $\mathbf{J}_{cm3}$ and $\mathbf{J}_{tip}$ are 6×9 geometric jacobian matrices relating the center of mass for the base, lower and upper crane link as well as the crane tip to the body-fixed frame of the vessel. The forces $\mathbf{F_{cm1}}$, $\mathbf{F}_{cm2}$, $\mathbf{F}_{cm3}$ and $\mathbf{F}_{tip}$ are 6×1 gravitational forces, and constructed in the following manner:

$$\mathbf{F}_{cm1} = \begin{bmatrix} 0 \\ 0 \\ m_{base}g \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{F}_{cm2} = \begin{bmatrix} 0 \\ 0 \\ m_{link1}g \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{F}_{cm3} = \begin{bmatrix} 0 \\ 0 \\ m_{link2}g \\ 0 \\ 0 \\ 0 \end{bmatrix} \quad \mathbf{F}_{tip} = \begin{bmatrix} 0 \\ 0 \\ (m_{spreader} + m_{cargo})g \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{111}$$

where m_{base}, m_{link1} and m_{link2} is the mass of the crane base as well as the lower and upper crane links. $m_{spreader}$, m_{cargo} is the mass of the spreader and cargo. g is the gravitational acceleration.

7.6 Wire and spreader

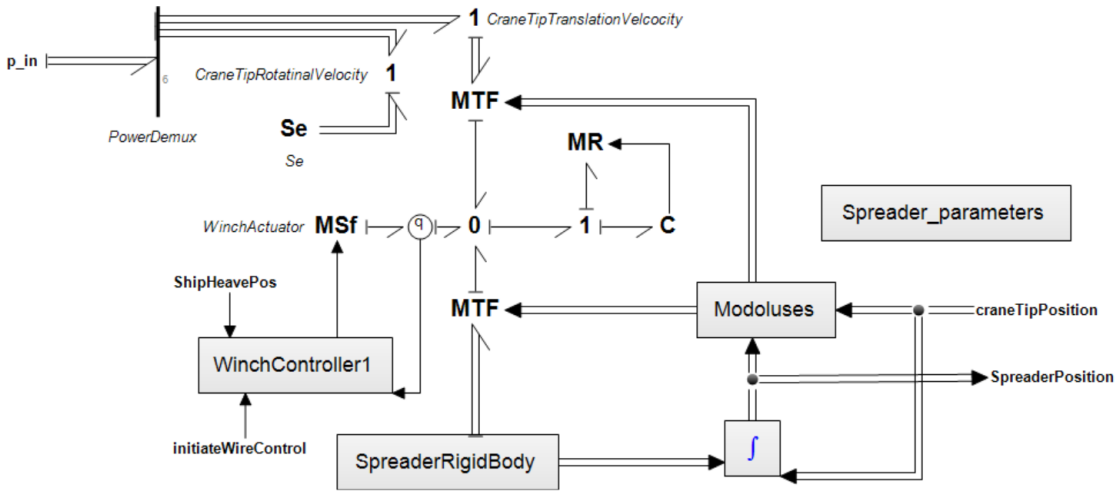


Figure 29: Bond graph model wire and spreader

The bond graph model shown in Figure 29 shows the wire model, the winch actuator, as well as the winch controller and the spreader which is modeled in the ***SpreaderRigidBody*** submodel.

The power demuxer shown in Figure 29 decompose a 6-dimensional power bond into two into 2 3-dimensional power bonds. The power bonds associated with rotational motion relative to the crane tip is set to zero, while the power bonds associated with translational motion is used in the wire and spreader model. The winch actuator is shown as a **MSf** element, and provides the winch velocity directly. The wire model is composed of two **MTF**-elements which are able to relate the wire displacement δ to the velocities at the top and bottom of the tether. The bond graph also contain a 1-junction which represent the resulting wire velocity $\dot{\delta}$, which is connected to a **C** and **MR** - element, which induce a damping and restoring force on the spring. The stiffness used in the **C** - element is found from the formula [6]

$$k = \frac{EA_w}{L_w} \tag{112}$$

where E is the young modulus of the wire, A_w is the cross-sectional area of the wire, and L_w is the extended wire length. In the case of the **MR**- element the damping coefficient defined as [6]

$$c = 2\zeta\sqrt{k_w m_w}. \tag{113}$$

The parameters c, ζ and m_w is the damping coefficient, relative damping, and wire-element mass respectively.

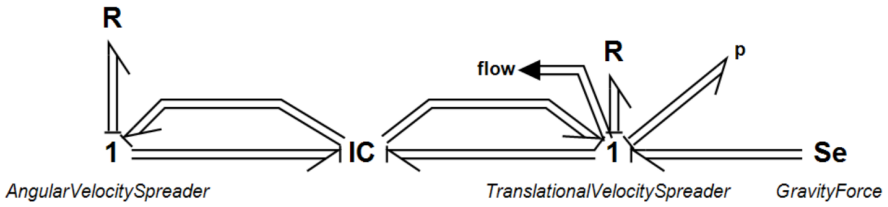


Figure 30: Rigid body bond graph model

Figure 30 show the rigid body model of the spreader, where the inertial and rigid body centripetal forces are implemented in the **IC** - field. The **R** represent friction forces from wind. The equation of motion for the spreader is given as:

$$\mathbf{M}\dot{\mathbf{v}} + \mathbf{C_{RB}}(\mathbf{v})\mathbf{v} + \mathbf{D}\mathbf{v} = \boldsymbol{\tau}. \tag{114}$$

Here $\mathbf{M}$ is the inertia tensor of the spreader, $\dot{\mathbf{v}}$ and $\mathbf{v}$ is the 6×1 acceleration and velocity vector of the spreader representing the translational and angular velocities of the spreader frame, while $\mathbf{C_{RB}}$ is the rigid body centripetal matrix. Lastly $\boldsymbol{\tau}$ is the external forces acting on the spreader.

7.7 Crane control and FSM

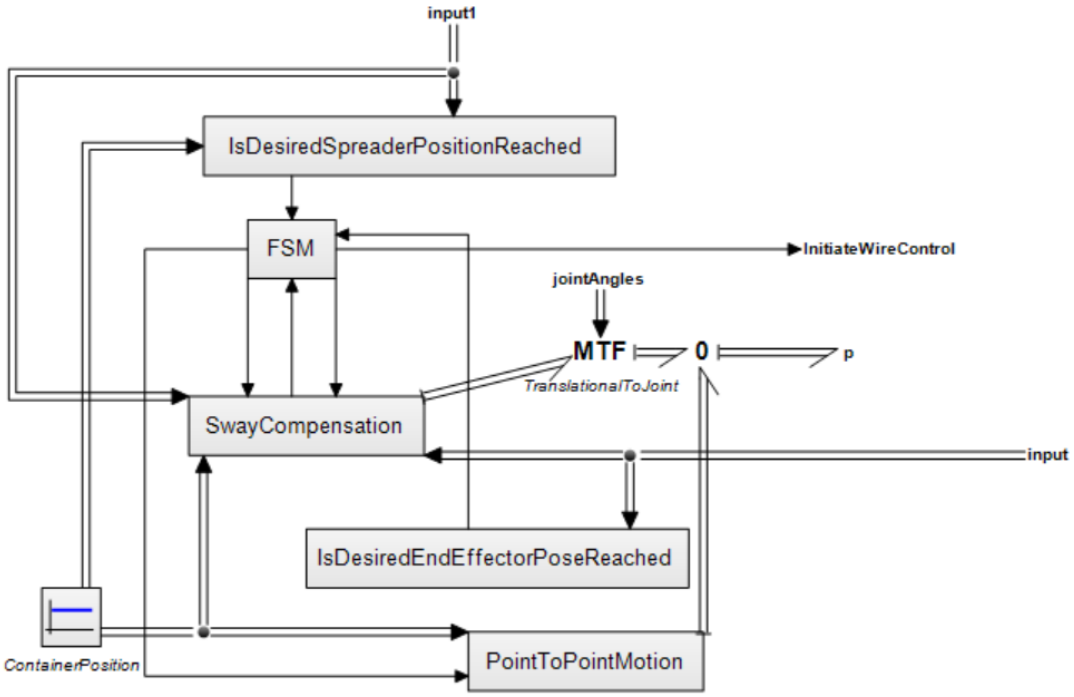


Figure 31: Crane control and FSM model

The implemented control system shown in Figure 31 takes in a desired container position in the world, and then computes the desired end-effector point, and how much the wire needs to be lowered in order for the spreader to reach the top of the container. The desired end-effector point is then sent to ***PointToPointMotion***, where the desired joint positions are computed by the use of inverse kinematics, and velocity profiles are computed. Once the end effector is sufficiently close to the desired end-effector point the sub-model ***IsDesiredEndEffectorPoseReached*** sends a boolean to the ***FSM*** to signal that a different task should begin. The system will then start sway-compensation and station-keeping of the crane-tip, and the sway controllers will be turned off after a predetermined amount of time has passed. The system will then start the last task which is wire-lowering and heave compensation. When the spreader is sufficently close to the top of the container, the submodel ***IsDesiredSpreaderPositionReached*** will send a boolean to the ***FSM*** and the system will enter the standby state.

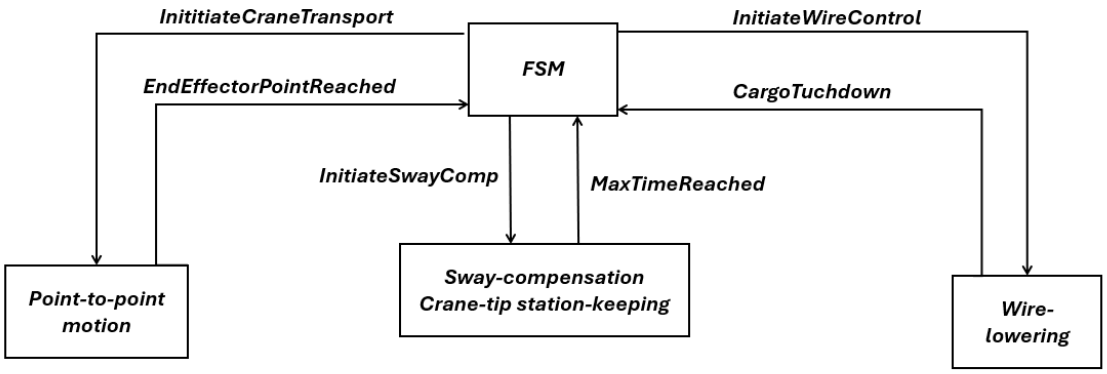


Figure 32: Block diagram of FSM implementation

How the FSM interfaces with each of the control algorithms can be seen in Figure 32. Each signal shown is a boolean. It is important to note that station-keeping of the crane-tip will still be running when the system lowers the wire.

7.8 Crane-tip station-keeping and sway-control

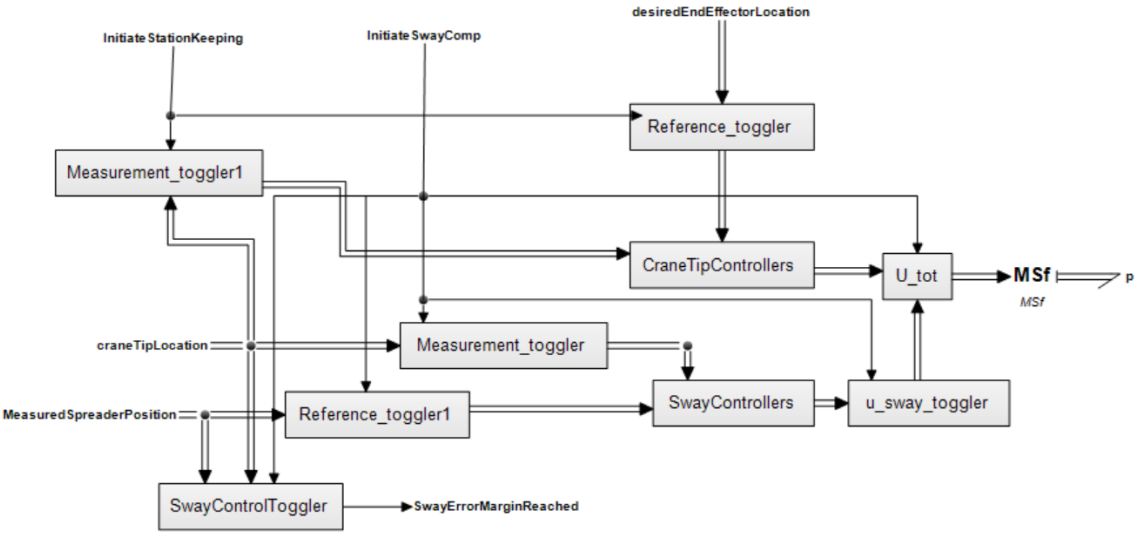


Figure 33: Sway and crane-tip control

Figure 33 shows how crane-tip-station-keeping and sway-compensation is implemented with regards to toggling from the FSM. The submodels ***Measurement_toggler***, ***Measurement_toggler1***, ***Reference_toggler***, ***Reference_toggler1*** all contain code that sets the reference and measurement to the controllers to be equal to zero when not in use. Additionally ***u_sway_toggler*** contain additional code to set the output of the sway controllers to zero when the state is toggled off. This was done as the controllers would sometimes deliver an output despite both the reference and measurement being set to zero. Lastly, the ***SwayControlToggler*** submodel is responsible for toggling the sway-control off when a sufficient time has passed. One can also see the submodels ***CraneTipControllers*** and ***SwayControllers***, which contain the controller for sway-control and

station-keeping.

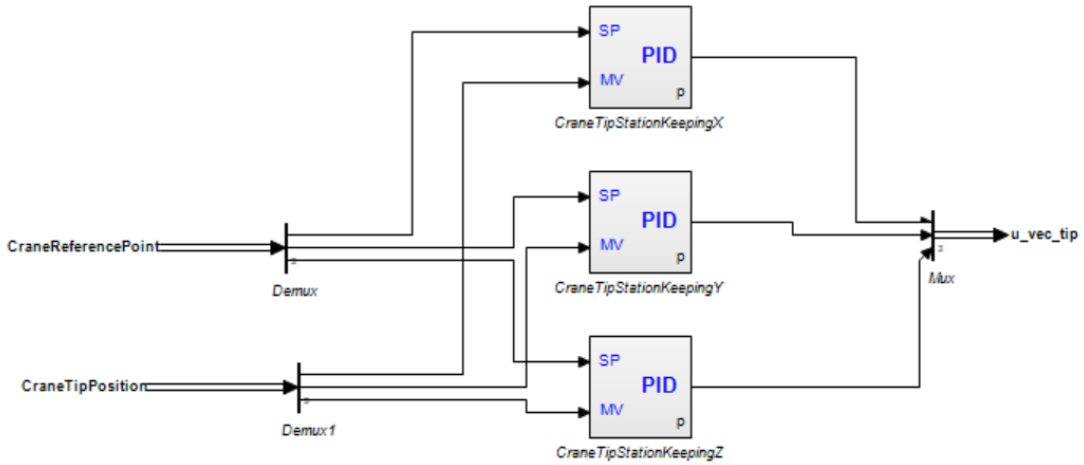


Figure 34: Implemented crane tip controllers

In Figure 34, it can be seen how the crane-tip station-keeping algorithm is implemented in *20-sim*. The control algorithm consists of 3 PID controllers, which perform station-keeping in each DOF of the crane. The PID controllers used are the same as the standard PID controllers from [24] with integral anti-windup added.

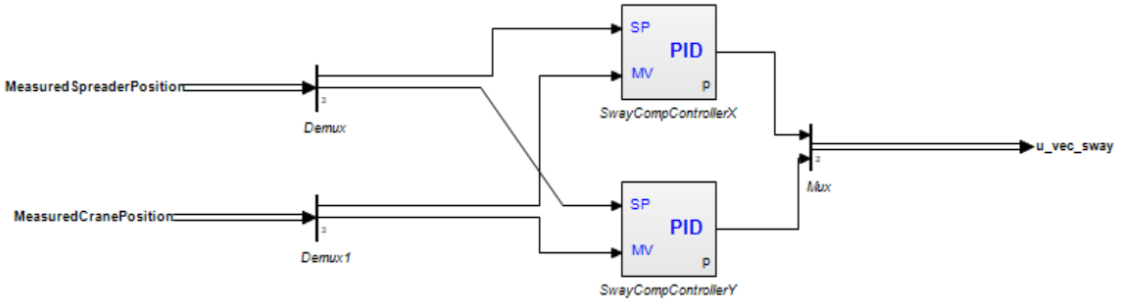


Figure 35: Implemented sway controllers

In Figure 35 one can see how the sway-controllers are implemented, and that it consists of 2 PID-controllers. One for x-direction, and one for y-direction. All the PID-controllers implemented in the thesis are similar with integral anti-windup.

7.8.1 Winch controller

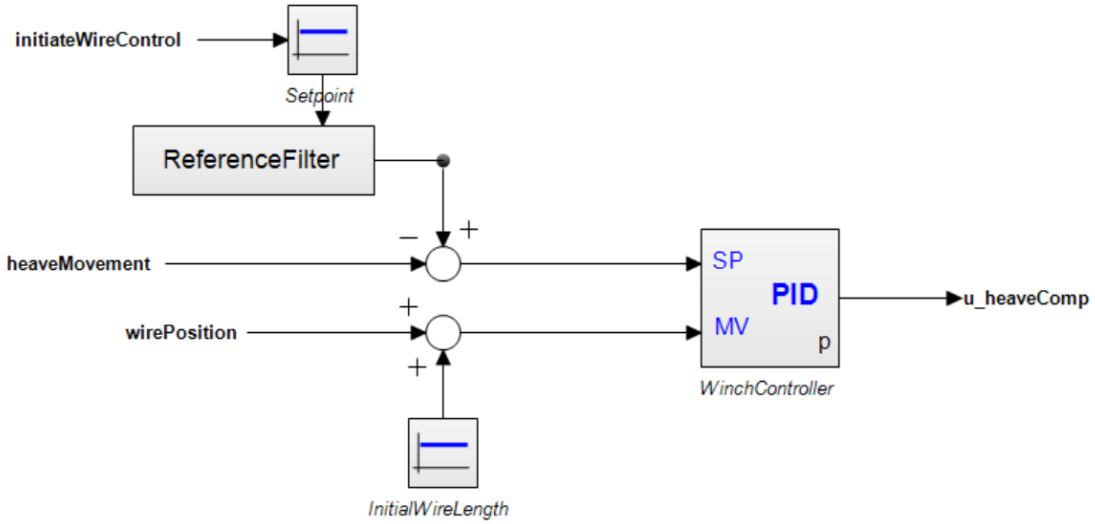


Figure 36: Implemented winch-controller and reference filter

Figure 36 shows how the winch controller and reference filter is connected. The signal for toggling the wire lowering task is also shown.

7.9 Implementation of point-to-point

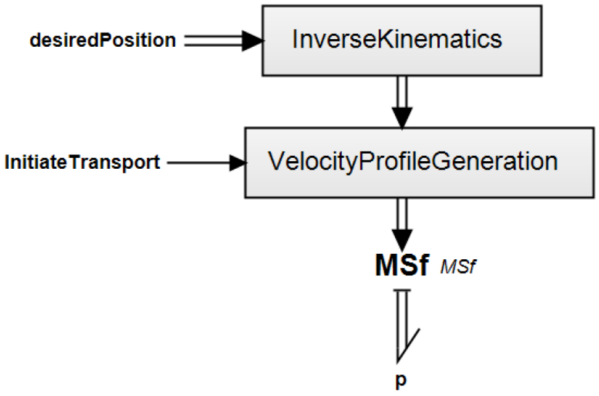


Figure 37: Implementation of point-to-point

Figure 37 shows that the implementation of the point-to-point motion algorithm is the same as the block diagram presented in Figure 18. The resulting velocities are then fed into a vectorial MSf element which sends the calculated velocities to the actuators.

8 Results and discussion

This chapter presents results that verify each of the algorithms presented in Section 6 work. Additionally, the performance of the whole control system while being supervised by the FSM will be presented. In the following cases the lowest joint will be referred to as the base joint. The joint of the lower arm will be referred to as joint1. The upper joint connecting the upper and lower arm will be referred to as joint2. This naming convention also corresponds to how each joint is named in Figure 15.

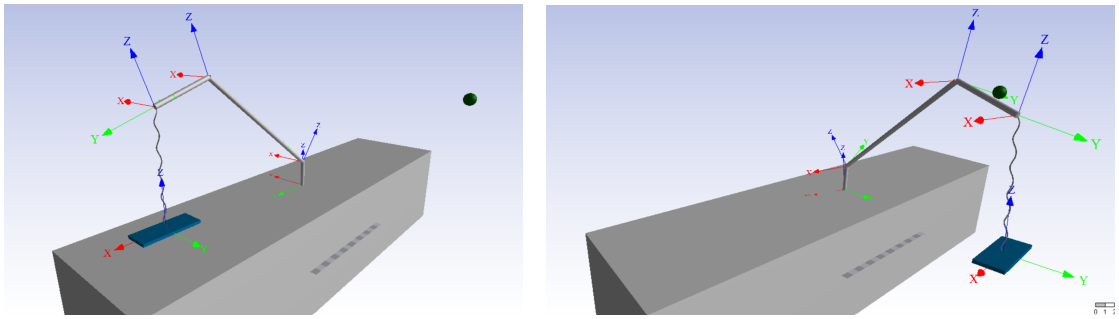
8.1 Result and discussion overview

The following chapter is composed of four subchapters. The first three chapters aim to verify that each of the control algorithms presented in Section 6 work by themselves, and the final subchapter investigates the integrated system under ideal conditions with no environmental forces.

- The first subchapter is ***Point-to-point point motion***, where the crane is made to reach a desired end-effector point by generating velocity profiles for each of the joints.
- The second subchapter is ***Sway-compensation and crane-tip station-keeping***, where the system performance with and without sway compensation is investigated by using an impulse force in x and y direction.
- The third subchapter is ***Wire control and heave compensation***. This chapter shows the performance of the wire-lowering algorithm with and without heave compensation when the Revolt vessel is exposed to a sinusoidal external force.
- Lastly the subchapter ***Integrated system [ideal conditions]*** aims to showcase how the system is able to attach the spreader mechanism to a container in the world by toggling between different modes of action. No environmental forces is included in this subchapter.

8.2 Point-to-point motion

The point-to-point motion algorithm consists of several steps. It first takes in a desired position in the world relative to the base frame, and uses inverse kinematics to generate the desired joint configurations. The algorithm then generates velocity profiles that the crane joints are made to follow in order to reach the desired end position. In order to verify that the algorithm worked as intended the crane was made to move from an initial position, to a desired end position.



(a) Initial crane position

(b) End crane position

Figure 38: Crane configurations before and after point-to-point motion. Desired end-effector position is shown as a green sphere.

Constraints	Base	Joint1	Joint2	Units
q_i	0	0.663	-0.995	$[rad]$
$\dot{q}_i$	0	0	0	$[rad/s]$
$\ddot{q}_i$	0.1	0.1	0.1	$[rad/s^2]$
q_f	1.326	0.702	-1.006	$[rad]$
$\dot{q}_f$	0	0	0	$[rad/s]$
$\ddot{q}_f$	-0.1	-0.1	-0.1	$[rad/s^2]$
t_f	10	10	10	$[s]$

Table 7: Constraints for each crane joint.

Here q_i, $\dot{q}_i$ and $\ddot{q}_i$ represent the intital joint position, velocity and acceleration, while q_f, $\dot{q}_f$ and $\ddot{q}_f$ are the end joint position, velocity and acceleration. Lastly t_f is the time the joint will take in order to reach the desired end configuration.

The following section will illustrate that while the crane joints are able to adhere to the constrains provided in Table 7, the algorithm does not care about the generated trajectory, as long as the constraints are adhered to.

8.2.1 Crane tip positions

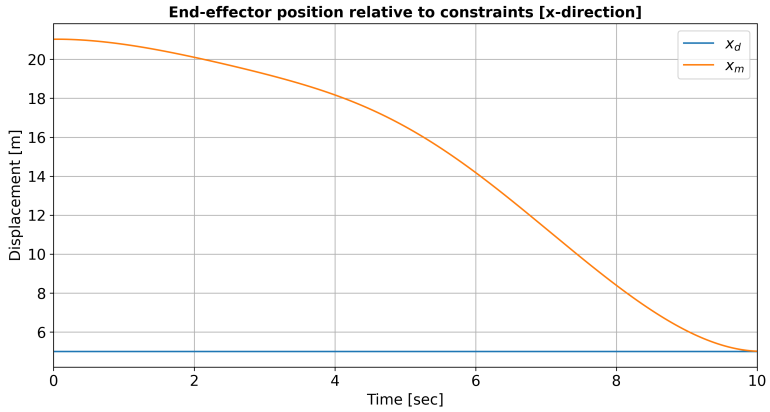


Figure 39: Here x_d and x_m is the desired and measured end-effector positions in x-direction.

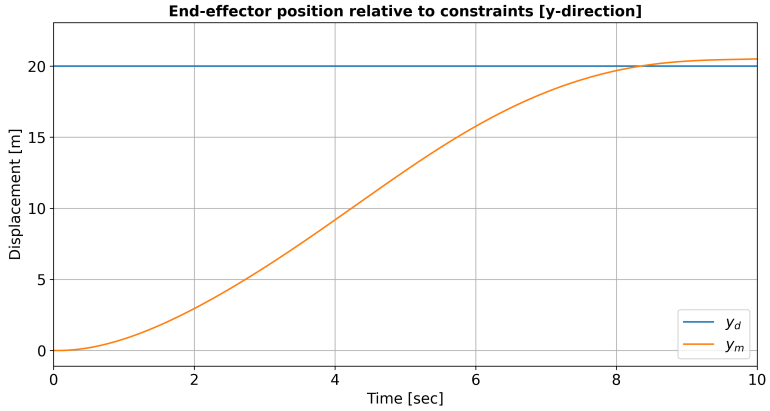


Figure 40: Here y_d and y_m is the desired and measured end-effector positions in y-direction.

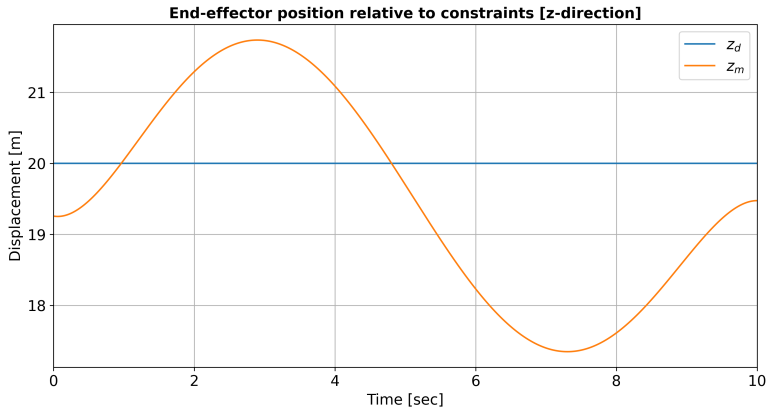


Figure 41: Here z_d and z_m is the desired and measured end-effector positions in z-direction.

The aim of the point-to-point motion algorithm is for the end-effector to reach the desired position in the world, and observing Figure 40 and Figure 41 one can conclude that the resulting end-effector has a significant offset relative to the desired position in the world in z and y direction. The offset, which is more apparent in Figure 41 than Figure 40, can be explained as the result of the ship rolling when the crane joints reach the desired rotation angles. Since the ship is rolling, it makes sense for the crane position to be lower in z, while slightly overextending in y. It can also be seen from Figure 39, that the x-position of the end-effector is able to reach the desired position. This is due to the pitch angle the vessel experiences when the crane is extended port-side would be negligible. Additionally the crane positions in Figure 41 is reminiscent of a sine curve. This behavior is unwanted, as traditionally one would want to take the most cost effective trajectory in order to reach the end goal, and highlights the limitations of the point-to-point motion algorithm.

8.2.2 Crane joint positions

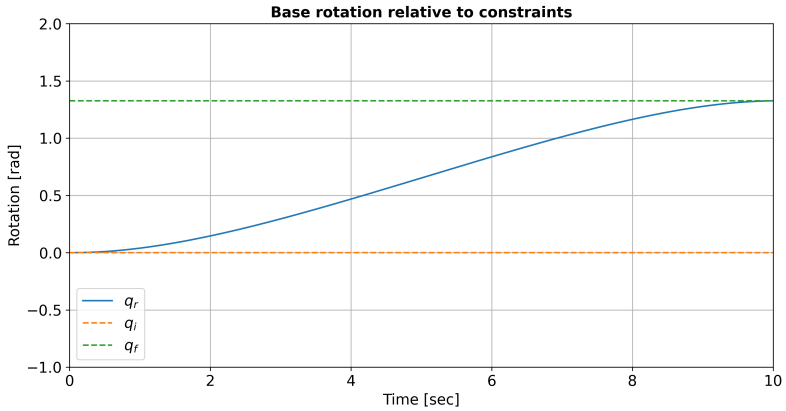


Figure 42: Here q_i and q_f denote the initial and end joint configurations for the base joint. q_r is the generated position reference of the base joint.

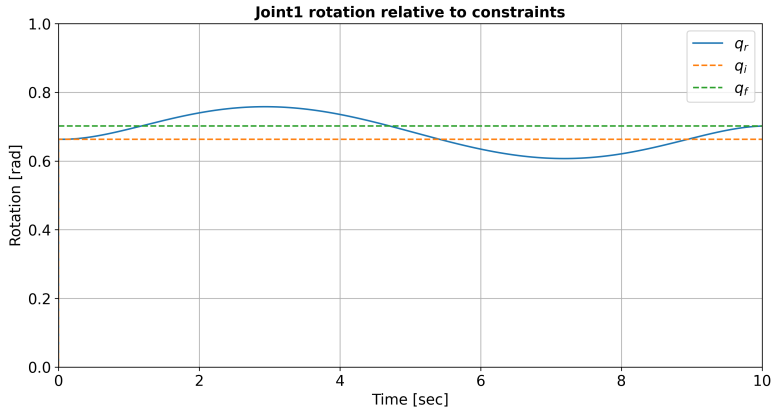


Figure 43: Here q_i and q_f denote the initial and end joint configurations for joint1. q_r is the generated position reference of joint1.

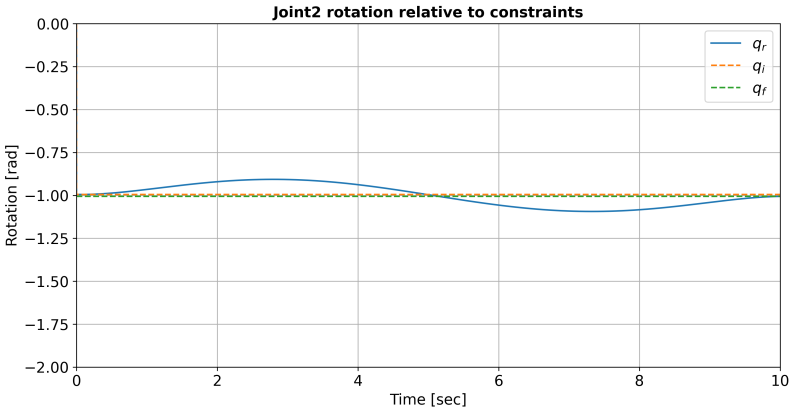


Figure 44: Here q_i and q_f denote the initial and end joint configurations for joint2. q_r is the generated position reference of joint2.

The Figure 42, Figure 43 and Figure 44 show how the crane joints move with respect to the constraints presented in Table 7. It can be seen that all crane joints adhere to the constraints, and is able to reach the desired orientations after 10 $[s]$. It should be noted that while the algorithm makes the crane reach a desired end position, it does not care about the path it takes to reach the desired end position. In an autonomous system, one would rather chain different set-points and generate trajectory curves based on what the situational awareness is able to detect.

8.2.3 Crane joint velocities compared to constraints

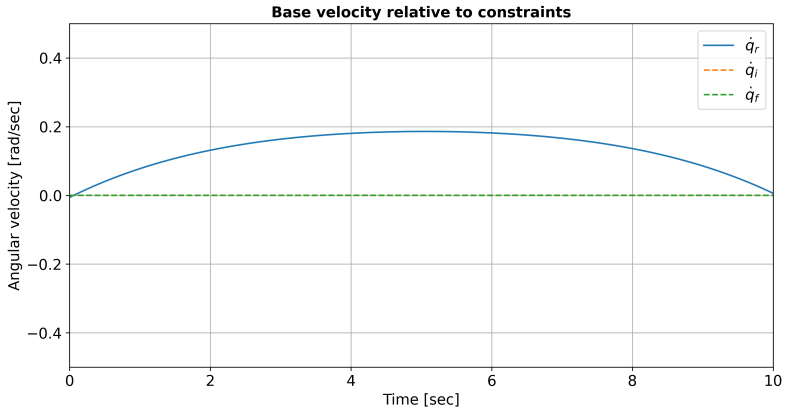


Figure 45: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for the base. $\dot{q}_r$ is the generated velocity reference of the base.

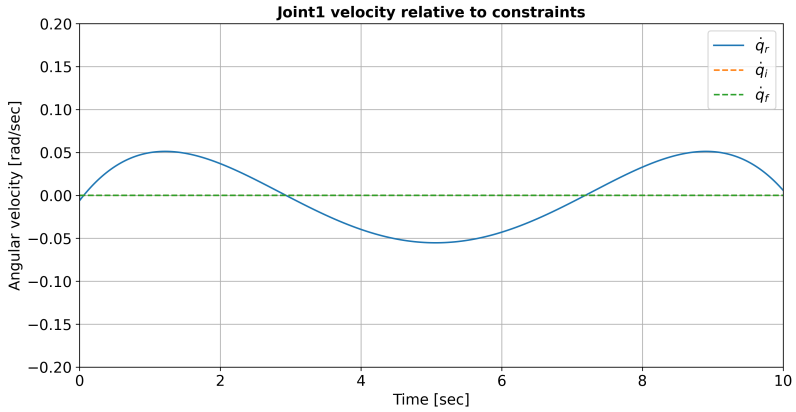


Figure 46: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for joint1. $\dot{q}_r$ is the generated velocity reference of joint1.

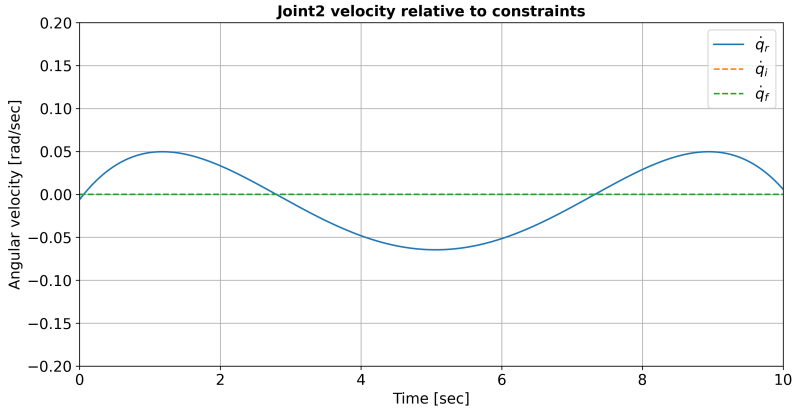


Figure 47: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for the joint2. $\dot{q}_r$ is the generated velocity reference of joint2.

Observing Figure 45, Figure 46 and Figure 47 one can see that the velocity profile generated adheres to the initial and end velocities presented in Table 7. It can be seen that the base velocity profile differs significantly from the velocity profiles of joint1 and joint2. This can be explained as a difference in constraints, as while all joints have the same constraints on the velocity and acceleration, the base joint needs to rotate over a larger span.

8.2.4 Crane joint accelerations compared to constraints

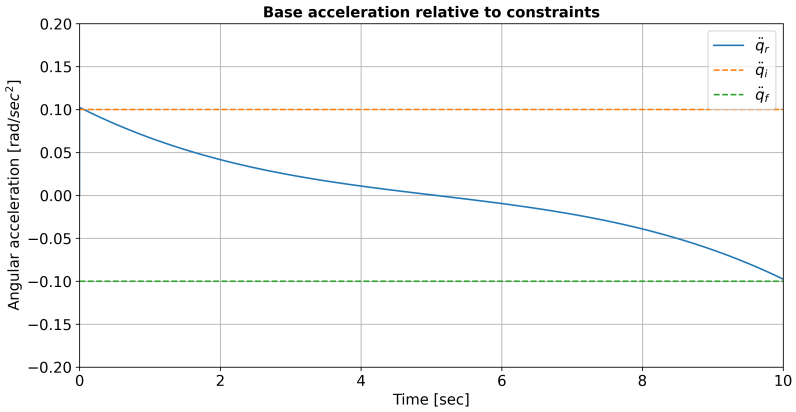


Figure 48: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for the base. $\ddot{q}_r$ is the generated acceleration reference of the base.

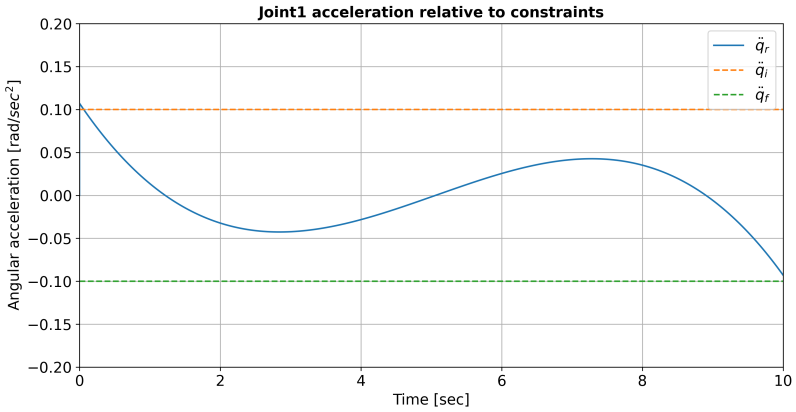


Figure 49: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for joint1. $\ddot{q}_r$ is the generated acceleration reference of joint1.

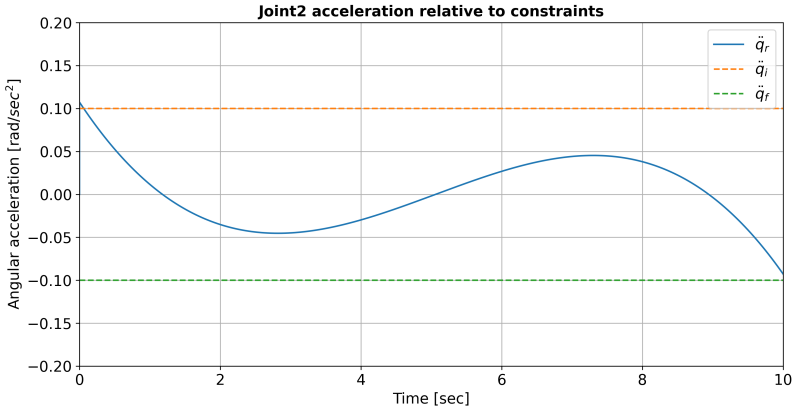
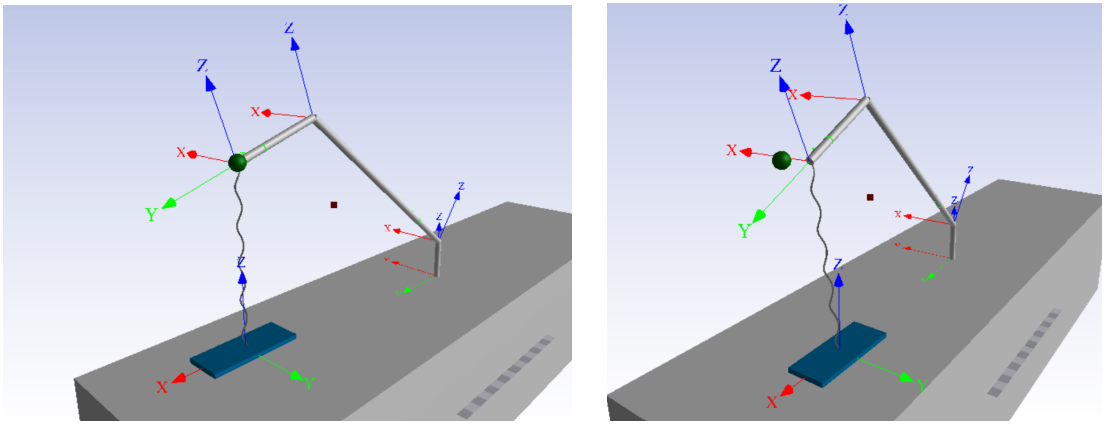


Figure 50: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for joint2. $\ddot{q}_r$ is the generated acceleration reference of joint2.

Lastly the joint accelerations presented in Figure 48, Figure 49 and Figure 50, can also be seen to adhere to the constraints presented in Table 7. One can also observe that the base joint has a different acceleration profile compared to the lower and upper joint. This may be due to the base joint needing to rotate over a larger span of angles compared to joint1 and joint2. It can be noted for an autonomous system utilizing point to point motion, it could be of interest to either adapt the constraints presented in Table 7 based on the angle span the joint need to travel, or utilize a different algorithm which would chain multiple points together while attempting to minimize the pendulum motions of the wire.

8.3 Sway-compensation and crane-tip station-keeping

Recall form section Section 6.3 that the sway compensation algorithm performs station-keeping of the crane tip while also minimizing the pendulum motions of the wire. The following section will illustrate that the algorithm is able to reduce the amplitude of the pendulum motions over time. In order to verify this, an impulse force was induced on the spreader mechanism in both x and y direction, and the response with and without sway compensation is presented.



(a) Initial crane position

(b) Spreader and crane after 4 sec

Figure 51: Crane and spreader positions shown at t=0 and t=4 sec. Desired end-effector position is shown as a green sphere.

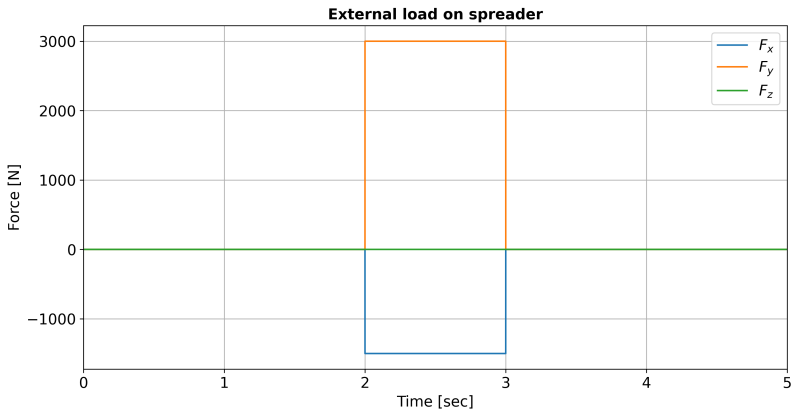


Figure 52: F_x, F_y and F_z is external forces in the x,y and z directions.

8.3.1 No sway compensation and stationary crane tip in x-direction

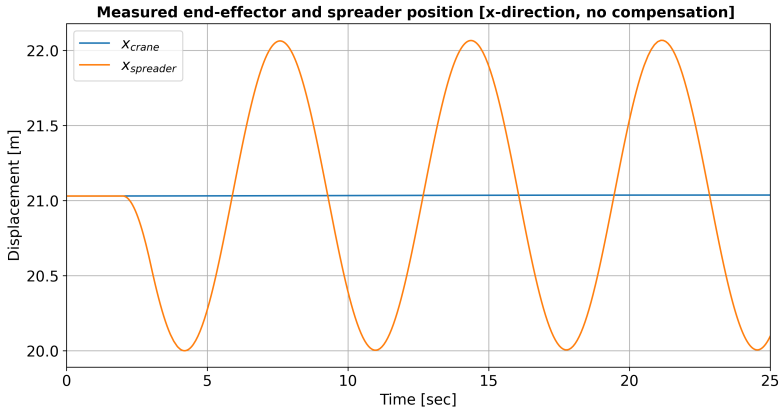


Figure 53: x_{crane} and $x_{spreader}$ is the x-positions of the crane-tip and the spreader mechanisms

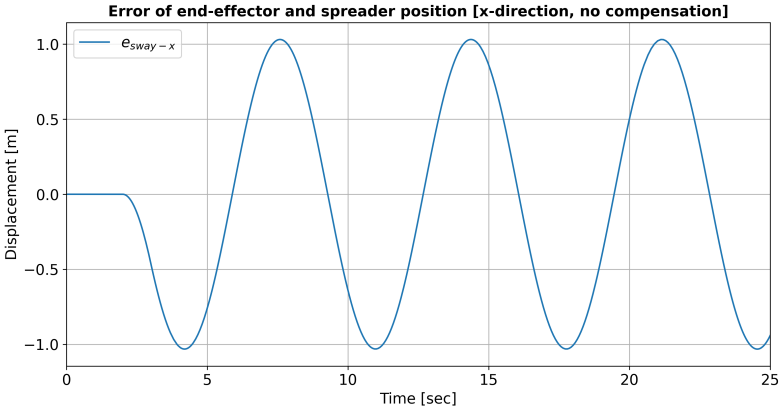


Figure 54: e_{sway-x} is the distance between the spreader and crane tip in x-direction.

The graphs in Figure 53, and Figure 54 show the spreader position relative to the stationary crane-tip as well as the error without any sway compensation. It can be seen that the pendulum motions do not seem to attenuate as the time increases.

8.3.2 No sway compensation and stationary crane tip in y-direction

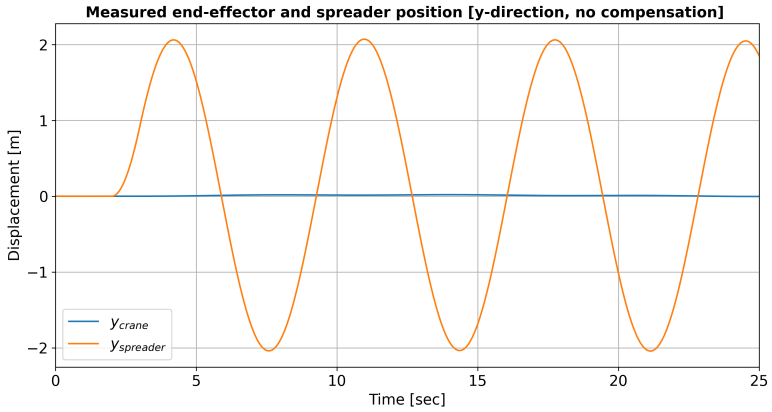


Figure 55: y_{crane} and $y_{spreader}$ are the y-positions of the crane-tip and the spreader mechanisms

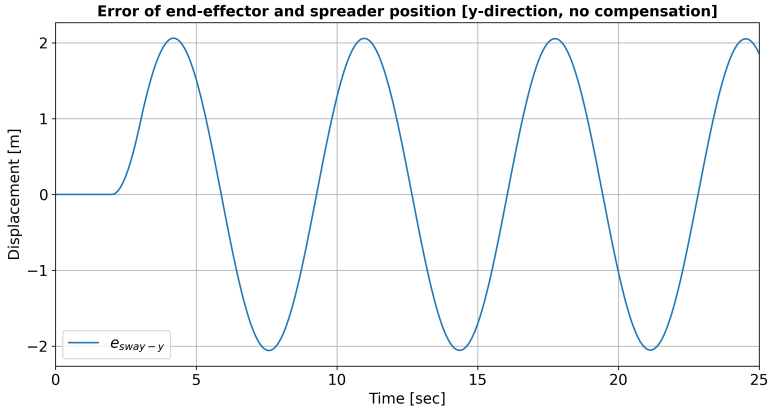


Figure 56: e_{sway-y} is the distance between the spreader and crane tip in y-direction.

Again it can be seen that without sway compensation, the y-position of the spreader in Figure 55 and the error shown in Figure 56 oscillate and does not seem to decrease over time.

8.3.3 Sway controller x-direction

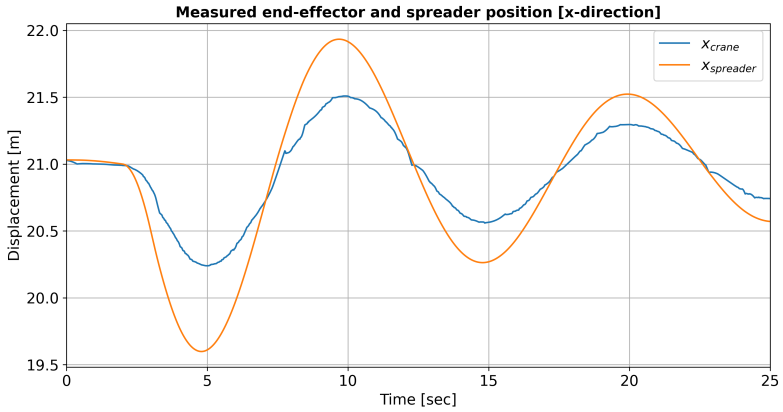


Figure 57: Here x_{crane} and $x_{spreader}$ are the x-positions of the crane-tip and the spreader mechanisms

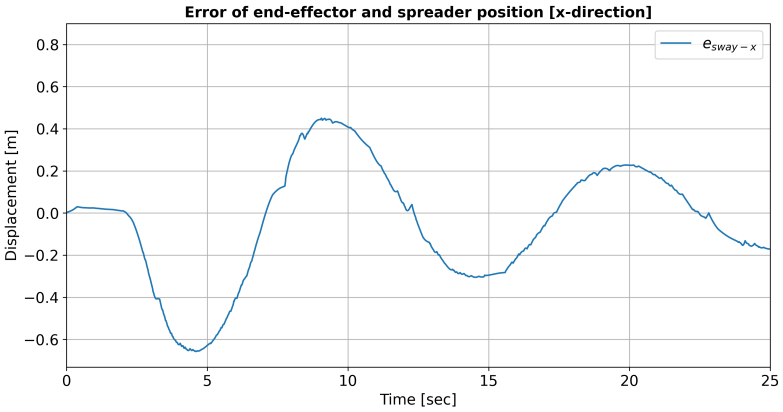


Figure 58: e_{sway-x} is the distance between the spreader and crane tip in x-direction.

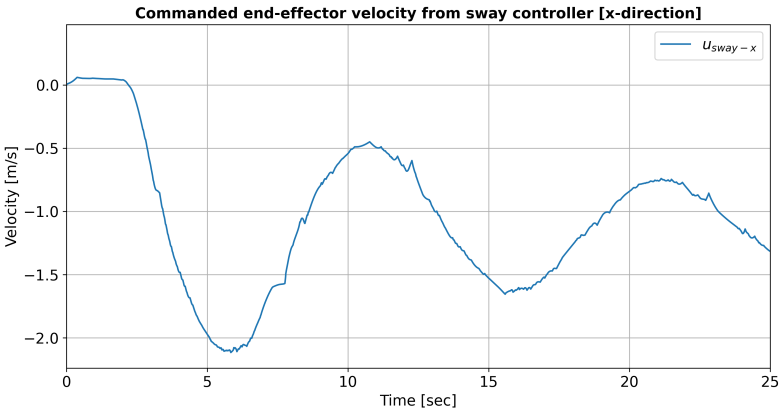


Figure 59: u_{sway-x} is the commanded end effector velocity from the sway controller in x direction.

Observing Figure 57 it can be seen that both the crane-tip as well as the spreader has a sinusoidal motion. Additionally it can be seen that the amplitude of motions attenuates as time increases. As a result of this the error shown in Figure 58 also decreases over time. Lastly the commanded crane tip-velocity from the sway controller shown in Figure 59 also decreases over time, which makes sense as the commanded crane tip velocity is a result of the error shown in Figure 58. However the commanded velocity in x-direction has several spikes, and ideally these would be removed from the control input as the joint actuators would experience unnecessary wear and tear from such rapid motions. A way to accomplish this would be to filter the control inputs through a low-pass filter. The spikes seen in the measured crane tip position could be a result of the controllers. The controllers provide a desired velocity to the actuators, and due to model simplifications it will be responded to instantly. This can explain why the crane tip is able to respond quickly enough to induce spikes in the measurements. However the controllers themselves should not induce noise to the control loop, and the noise generated from the controllers may be due to numerical errors that originate from the solver. It is possible by reducing the timestep on the solver would give more desirable results.

8.3.4 Sway controller y-direction

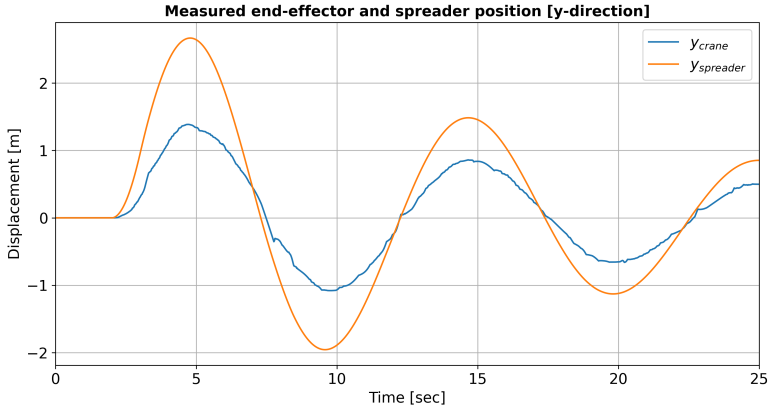


Figure 60: Here y_{crane} and $y_{spreader}$ are the y-positions of the crane-tip and the spreader mechanisms

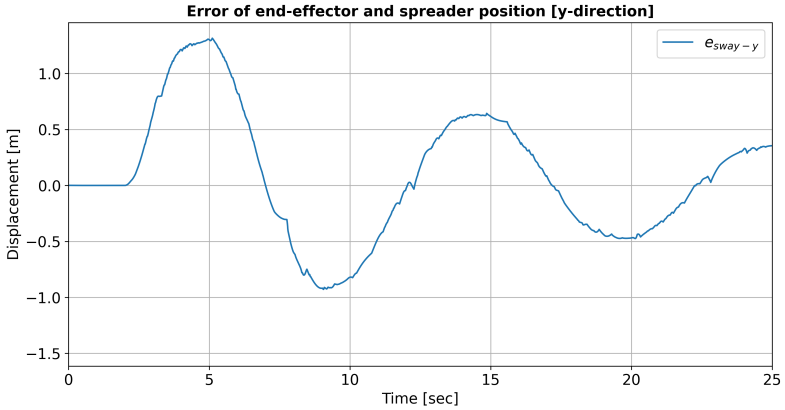


Figure 61: e_{sway-y} is the distance between the spreader and crane tip in y-direction.

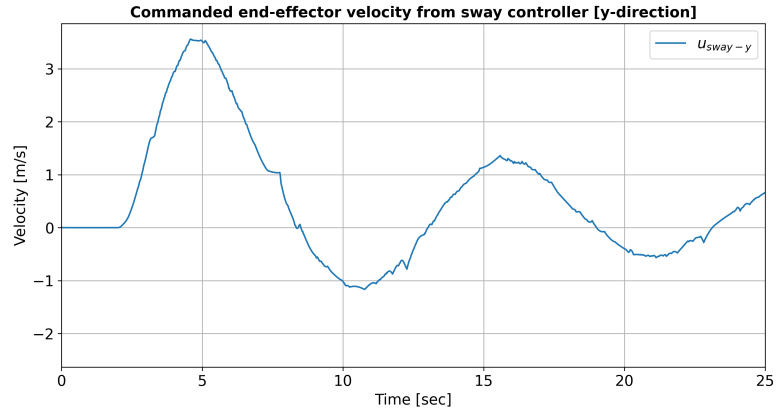


Figure 62: u_{sway-y} is the commanded end effector velocity from the sway controller in y direction.

Again it can be seen from the spreader and crane tip motions shown in Figure 60, that they decrease over time. Again this results in the error shown in Figure 61 to also decrease over time. Lastly the commanded crane-tip velocity from the end-effector shown in Figure 62 also decreases over time, and have the same spikes as discussed in Section 8.3.3.

8.3.5 Station-keeping x-direction

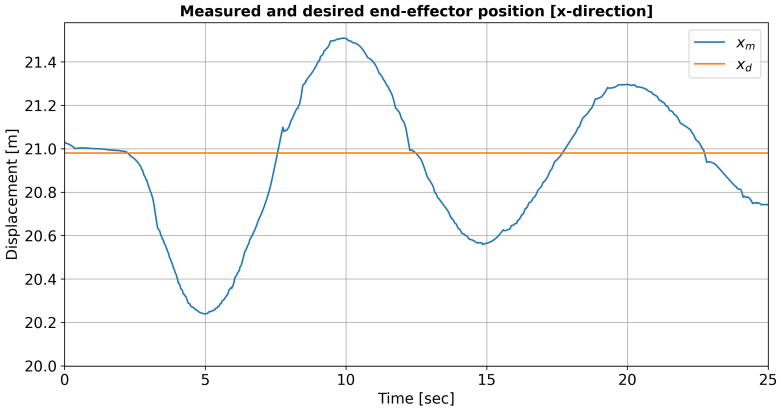


Figure 63: Here x_m and x_d is the measured and desired crane positions in x-direction.

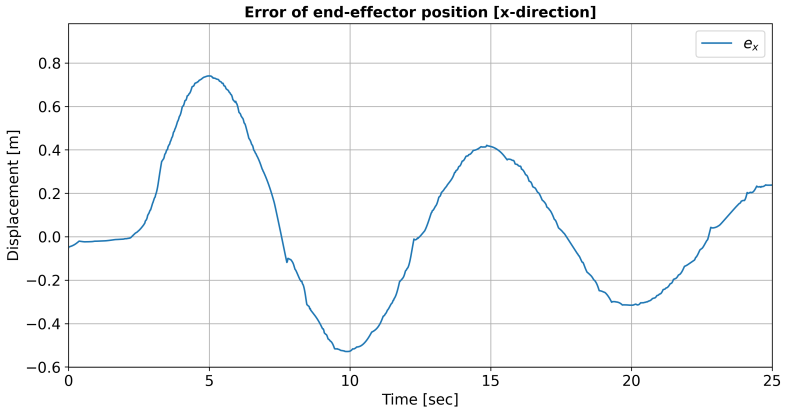


Figure 64: Here e_x is the error between the desired and measured crane tip position in x-direction.

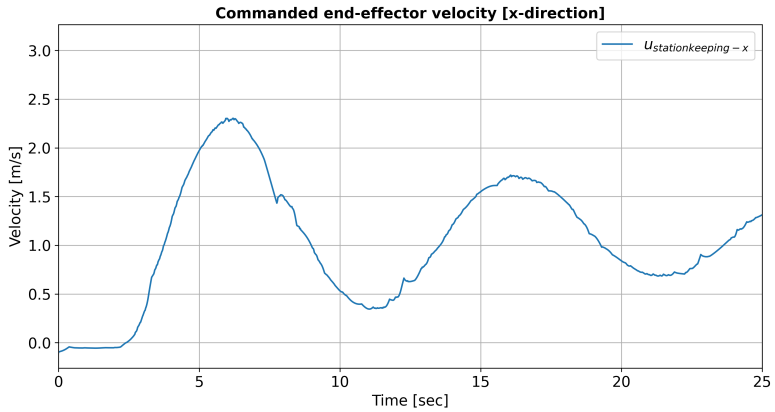


Figure 65: Here $u_{stationkeeping-x}$ is the commanded crane-tip velocity in x direction delivered from the station-keeping controller.

Observing the crane-tip reference and measurement in Figure 63 it can be seen that the crane tip position converges towards the reference as time increases. This results in the error shown in Figure 64 to decrease over time. However the commanded crane-tip velocity in Figure 65 seems to stabilize around a nonzero value. This could be due to the staitonkeeping controller competing against the sway controller which has the crane-tip follow the spreader mechanism.

8.3.6 Station-keeping y-direction

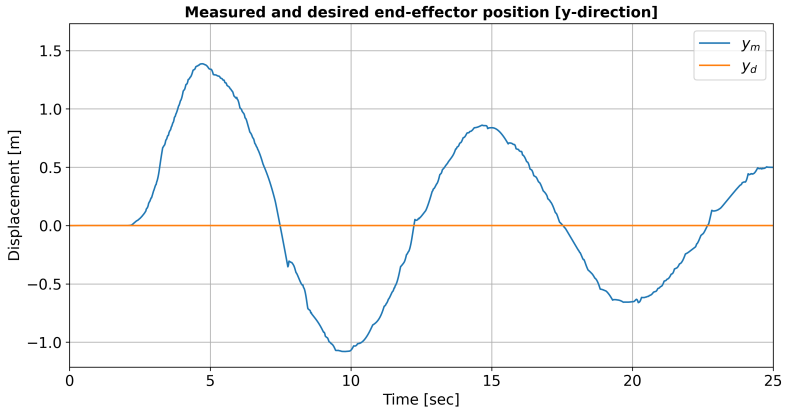


Figure 66: Here y_m and y_d is the measured and desired crane positions in y direction.

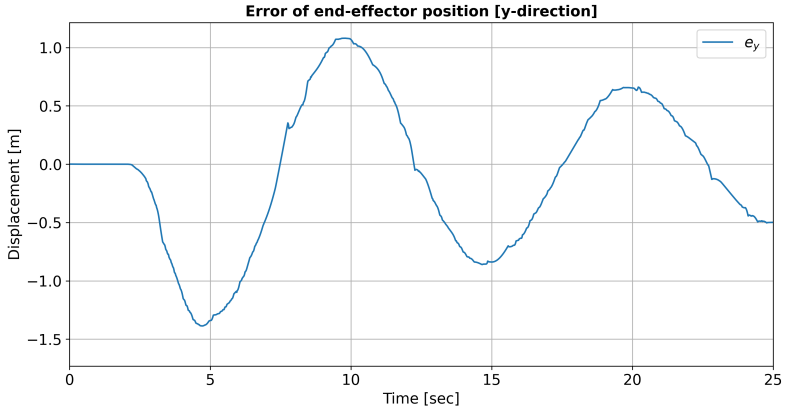


Figure 67: Here e_y is the error between the desired and measured crane tip position in y -direction.

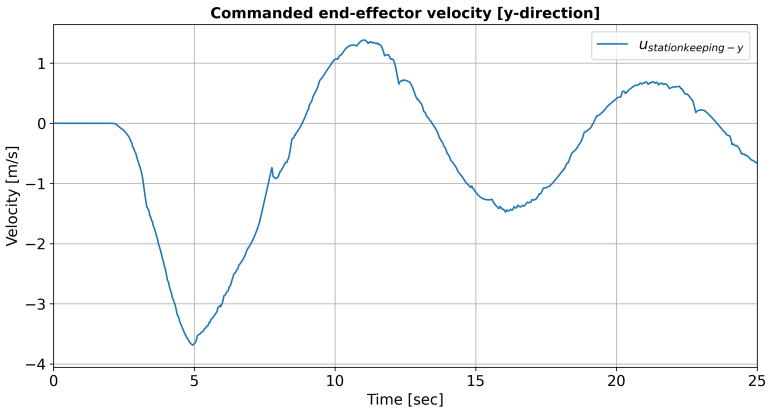


Figure 68: Here $u_{stationkeeping-y}$ is the commanded crane-tip velocity in y direction delivered from the station-keeping controller.

The results shown in Figure 66, Figure 67 and Figure 68 show the same behavior as discussed in Section 8.3.5 with an error that decreases over time.

8.3.7 Station-keeping z-direction

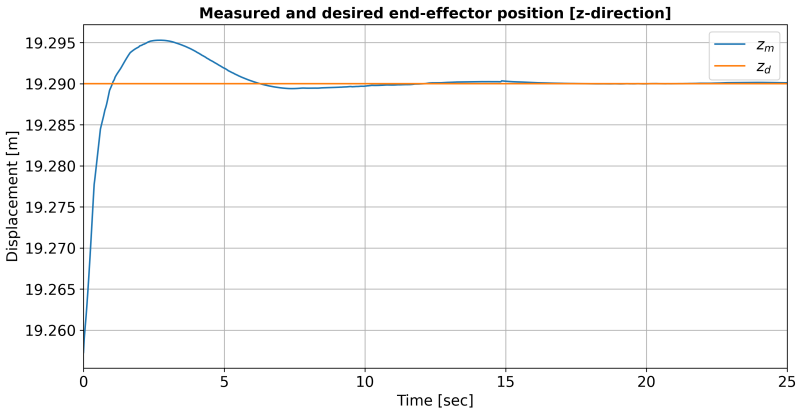


Figure 69: Here z_m and z_d is the measured and desired crane positions in z direction.

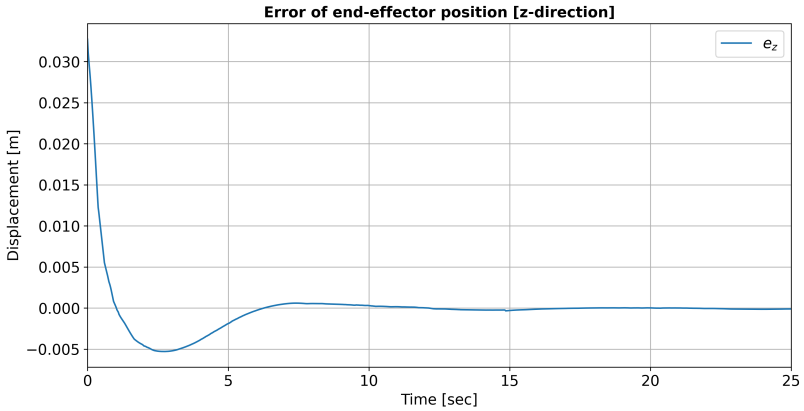


Figure 70: Here e_z is the error between the desired and measured crane tip position in z -direction.

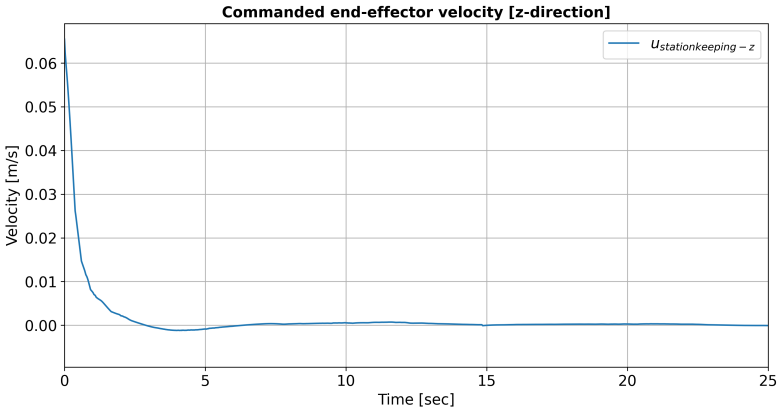
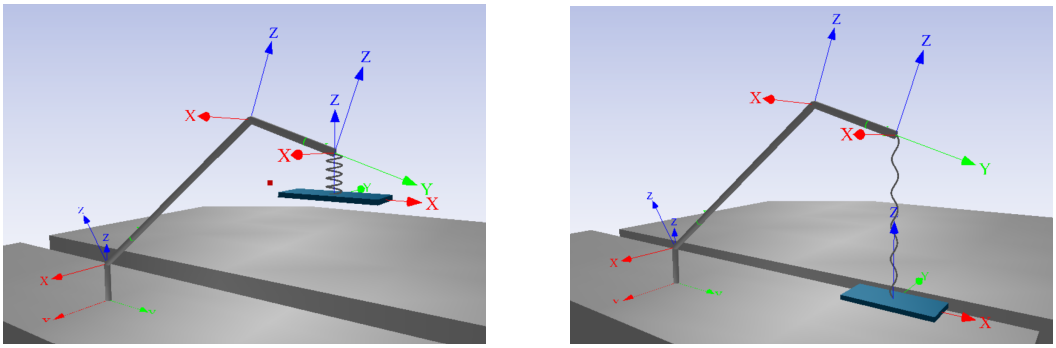


Figure 71: Here $u_{stationkeeping-z}$ is the commanded crane-tip velocity in z direction delivered from the station-keeping controller.

The controller in z-direction can be seen to reach the desired end effector position after approximately 6 seconds, as shown in figure Figure 69. This in turn gives an error and control input that converges to 0 as shown in Figure 70 and Figure 71.

8.4 Wire control and heave compensation

Recall from Section 6.4, that the wire control algorithm utilizes a reference filter that generates the desired wire position for the winch to follow. The heave compensation is implemented by subtracting the heave motions of the vessel in the final reference. This results in a time varying reference that the winch controller can follow in order to remove heave motions from the spreader. In order to verify wire-control and heave compensation two cases will be discussed, namely wire lowering with and without heave compensation.



(a) Initial spreader position. Wire-length = 2.5 $[m]$ (b) End spreader position. Wire-length = 10 $[m]$

Figure 72: Spreader position before and after wire control

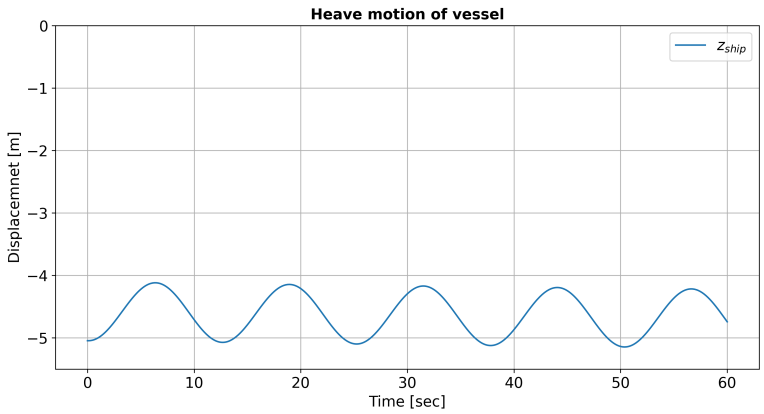


Figure 73: Heave motion of vessel for both cases, z_{ship} is the vertical movements of the ship.

The initial and end spreader positions are shown in Figure 72, and the heave motion the vessel experiences in both cases is presented in Figure 73. It is worth noting that the measured wire position is measured relative to the initial wire length which is 2.5 $[m]$.

8.4.1 Wire lowering without heave-compensation

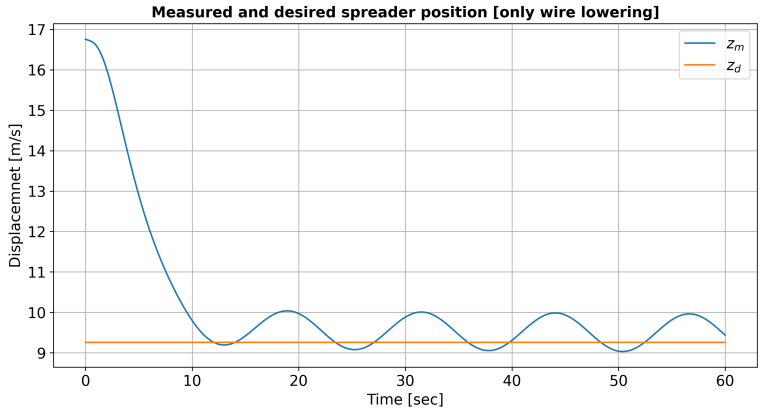


Figure 74: z_m and z_d is the measured and desired spreader positions in z-direction.

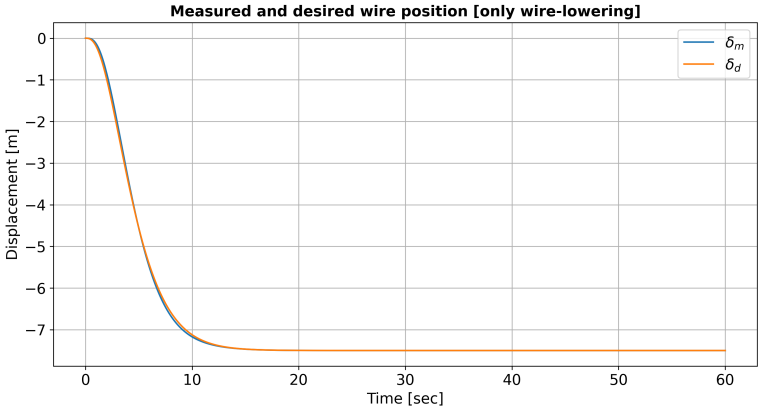


Figure 75: δ_m and δ_d is the measured and desired wire lengths relative to an initial wire length of 2.5 $[m]$.

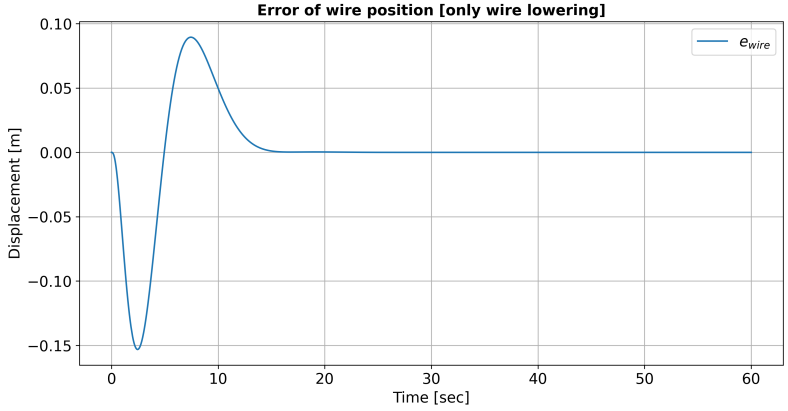


Figure 76: e_{wire} is the error between the measured and desired wire lengths.

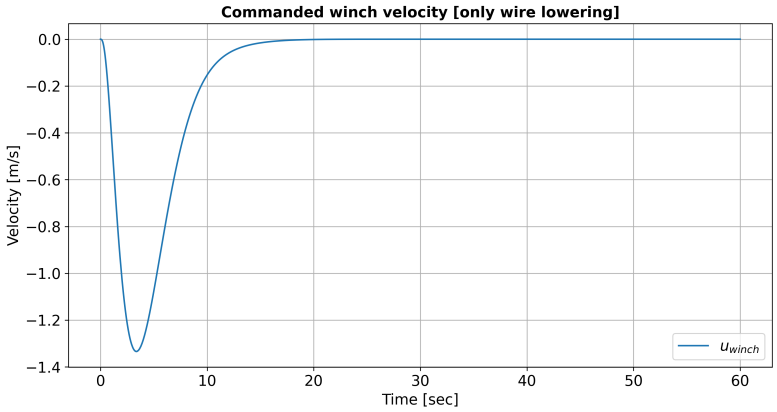


Figure 77: u_{winch} is the commanded winch velocity.

Observing the measured compared to the desired spreader position shown in Figure 74, it can be seen that the vertical spreader position oscillates with an amplitude of approximately 0.5 *[m]*, and is not able to reach the wanted spreader position as time increases. Additionally, it can be seen that with only wire lowering, the system is able to track the generated reference as can be seen in Figure 75. This in turn gives an error that converges to 0 as shown in Figure 76. Lastly it can be seen that the commanded winch velocity shown in Figure 77 also converges to 0 as the wire converges to the reference.

8.4.2 Wire lowering with heave-compensation

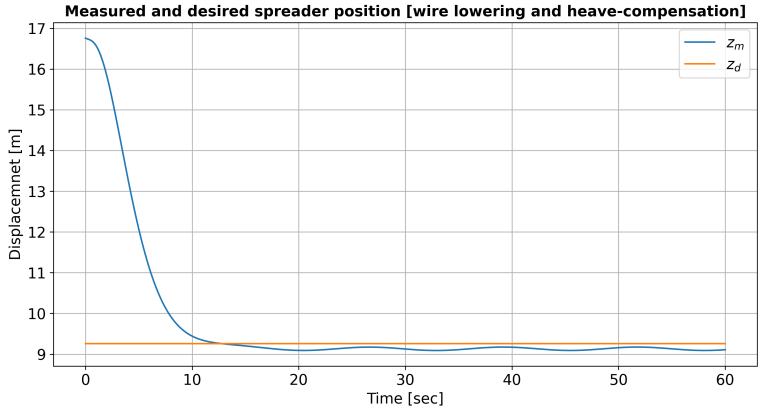


Figure 78: z_m and z_d is the measured and desired spreader positions in z-direction.

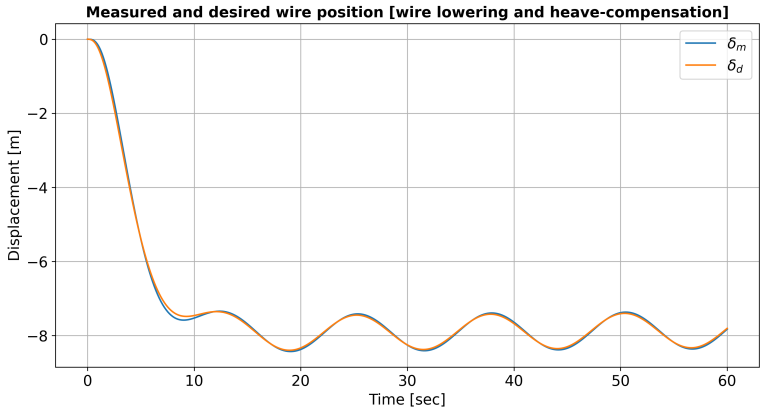


Figure 79: δ_m and δ_d is the measured and desired wire lengths relative to an initial wire length of 2.5 $[m]$.

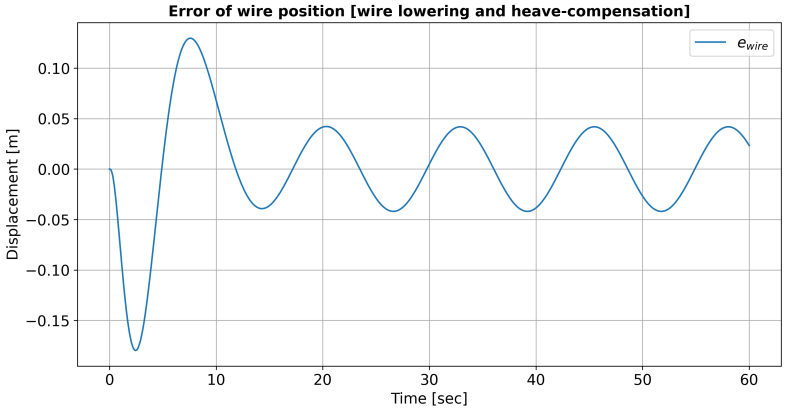


Figure 80: e_{wire} is the error between the measured and desired wire lengths

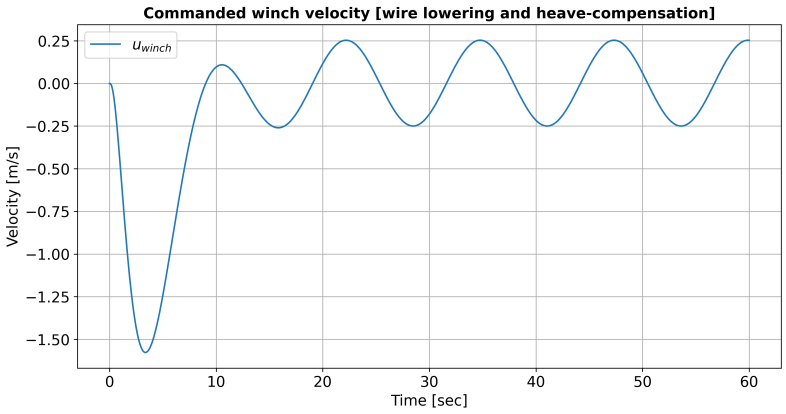


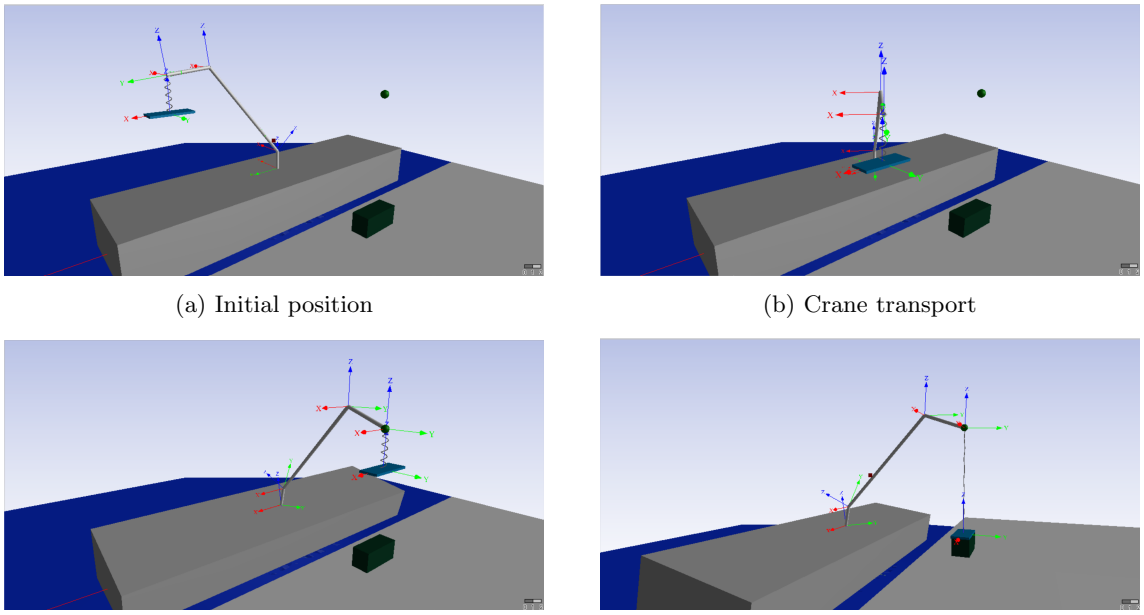
Figure 81: u_{winch} is the commanded winch velocity while taking heave motions into account.

Inspecting the spreader position relative to the wanted reference shown in Figure 78, one can see that the error is able to stay close to the reference, and the earlier oscillations have now mostly died

out. However there is an offset between the end spreader position and the desired reference. This could be due to the resulting pitch motion the vessel experiences due to the crane and spreader weight. This response can be explained by observing the measured and desired wire position shown in Figure 79. Here it can be seen that the new reference is able to take the heave motions of the vessel into account, which in turn cancels out most of the heave motions. The error between the desired and measured wire position shown in Figure 80 shows that the controller is able to respond to the reference reasonably well. Lastly it can be seen that the commanded winch velocity shown in Figure 81 responds to the oscillations in the reference model.

8.5 Integrated system [ideal conditions]

The integrated system is able to move the spreader to a given container position in the world. This is done by first utilizing point-to-point motion to reach the approximate position of the desired-end effector point. Once the end-effector is in close enough proximity, the FSM sets the velocity profiles generated from point-to-point motion to 0, and toggles sway-compensation and station-keeping. Once 15 $[s]$ pass the FSM then toggles off sway-compensation, and enters the wire-control state. It is worth noting that station-keeping of the end-effector is turned on both during the sway-compensation as well as the wire-lowering phase. In the following case there are no external forces, and the system performance under ideal conditions are investigated. Additionally, friction forces acting on the spreader from wind is turned off in this case.



(a) Initial position

(b) Crane transport

(c) End-effector position reached

(d) Spreader hits top of container

Figure 82: Screenshots of simulated case. Container and desired end-effector point are shown as a box, and green sphere respectively.

In Figure 82 one can see the different phases of simulated case. In Figure 82a one can see the initial position of the spreader and end-effector which initializes at the stern of Revolt. In Figure 82b one can see the system while performing point-to-point motion. Lastly in Figure 82c and Figure 82d one can see the crane reaching the desired position, and the spreader landing at the top of the container.

Constraints	Base	Joint1	Joint2	Units
q_i	0	0.663	-0.995	$[rad]$
$\dot{q}_i$	0	0	0	$[rad/s]$
$\ddot{q}_i$	0.1	0.1	0.1	$[rad/s^2]$
q_f	1.326	0.702	-1.006	$[rad]$
$\dot{q}_f$	0	0	0	$[rad/s]$
$\ddot{q}_f$	-0.05	-0.1	-0.1	$[rad/s^2]$
t_f	15	10	10	$[s]$

Table 8: Constraints for each crane joint in integrated case.

Table 8 shows the constraints fed into pont-to-point motion. The constraints are mostly the same as before, but now the desired end-acceleration $\ddot{q}_f$ and t_f of the base joint is set to -0.05 $[rad/s^2]$ and 15 $[s]$, as a smaller acceleration would give less pendulum motions.

8.5.1 FSM and state toggling

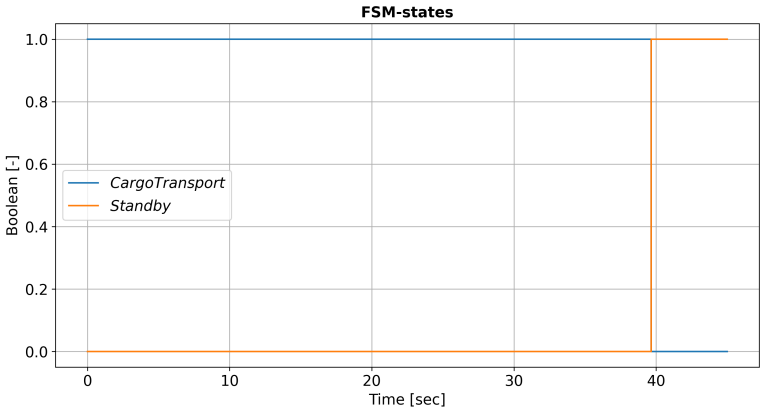


Figure 83: Graph showing when standby and cargo transport state is toggled.

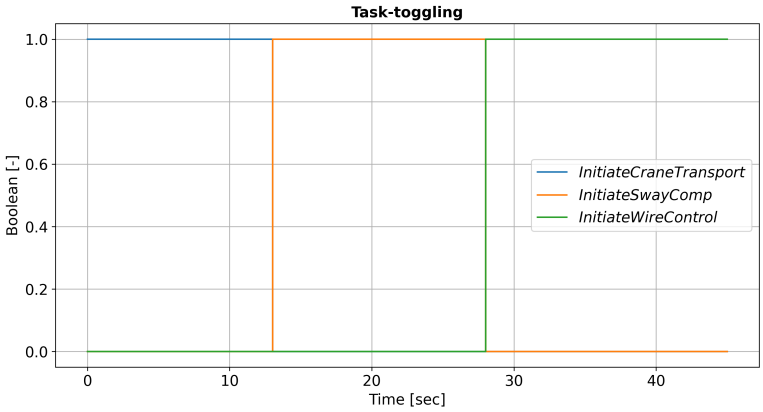


Figure 84: Graph showing when crane-transport,sway-compensation and wire-control are toggled

From Figure 83 and Figure 84 showing the two main modes of operation of the crane system, as well as the task toggling it can be observed that the system is able to successfully toggle between different modes of operation.

8.5.2 Desired compared to desired end-effector-position

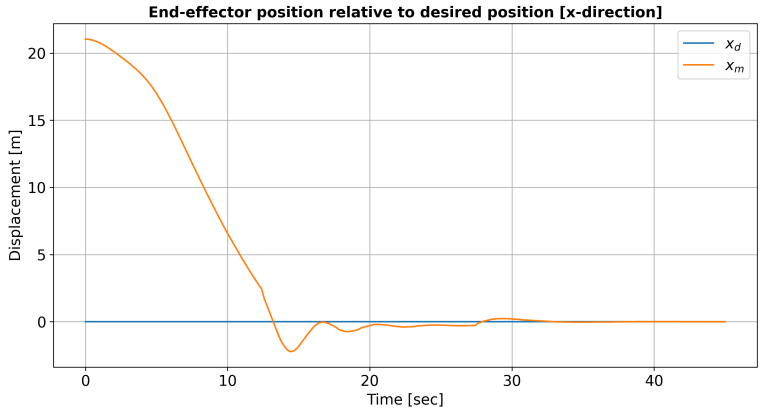


Figure 85: x_d and x_m is the desired and measured end-effector position in x-direction

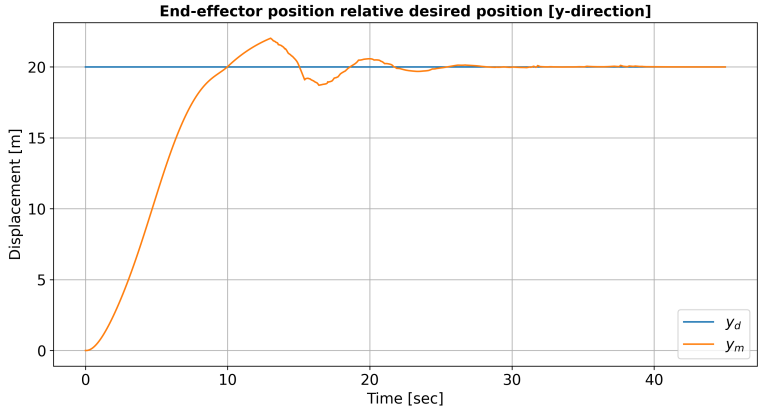


Figure 86: y_d and y_m is the desired and measured end-effector position in y-direction

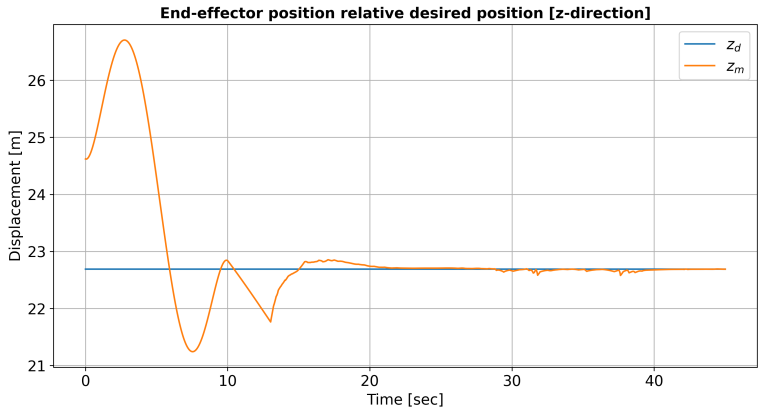


Figure 87: z_d and z_m is the desired and measured end-effector position in z-direction

The positions of the end-effector relative to the desired position in x, y and z direction shown in Figure 85, Figure 86 and Figure 87 show that the end-effector is able to converge towards the desired position. It can be seen that around 12 $[s]$ the crane-tip position undergoes a change in response in all three figures. This is due to the system changing between point-to-point motion and the crane tip station-keeping algorithms, as the end-effector is deemed sufficiently close to the desired point to change modes from crane-transportation to sway-compensation and station-keeping as can be seen in Figure 84.

8.5.3 Spreader position compared to desired container position

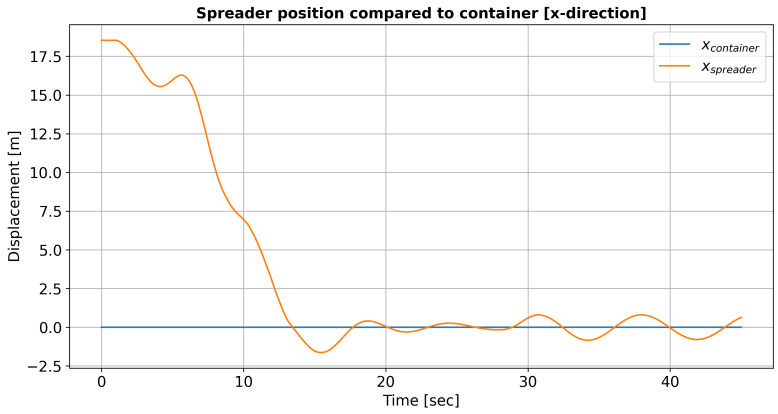


Figure 88: $x_{spreader}$ and $x_{container}$ is the position of the spreader and container in x-direction

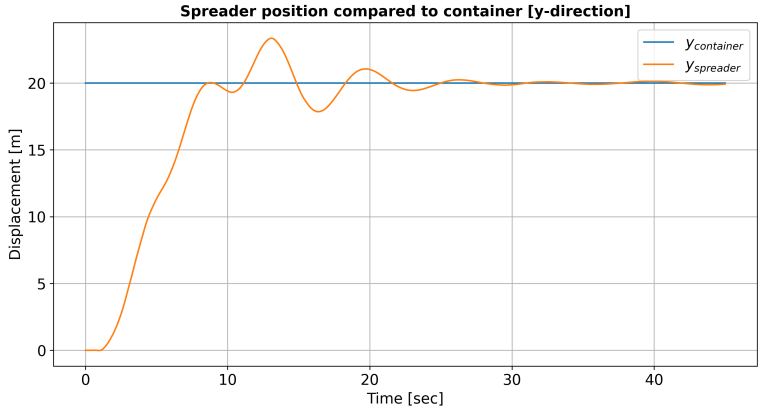


Figure 89: $y_{spreader}$ and $y_{container}$ is the position of the spreader and container in y-direction

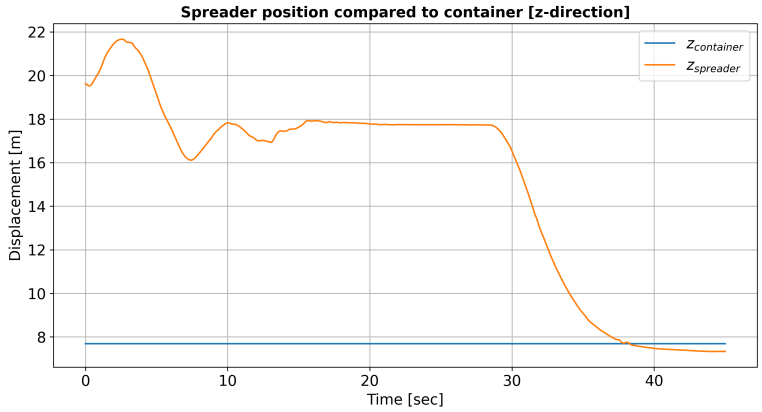


Figure 90: $z_{spreader}$ and $z_{container}$ is the position of the spreader and container in z-direction

The figures Figure 88, Figure 89 and Figure 90 show the spreader position compared to the desired container position in x,y and z direction respectively. It can be seen that at 45 $[s]$ the spreader is able to converge towards the container position in z-direction, but overshoots by approximatly 10 $[cm]$. This overshoot may stem from the reference filter itself as it will later be seen that the crane-tip and winch controllers are able to reach their references. Additionally, it still oscillates in x, and y -direction. This could possibly be improved by not stopping the sway compensation algorithm even during the wire-lowering phase.

8.5.4 Point-to-point motion performance [base]

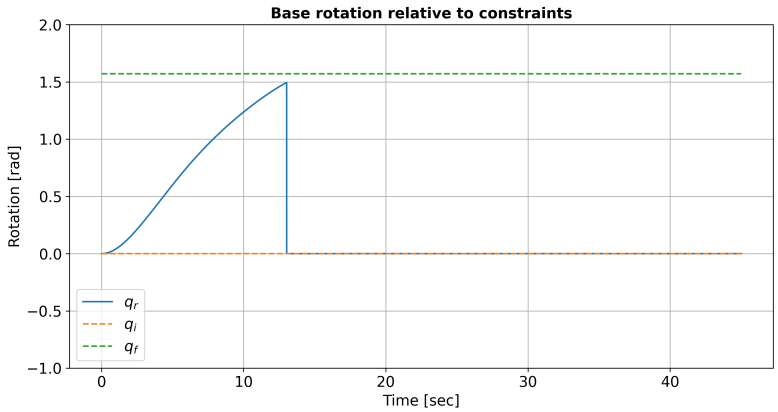


Figure 91: Here q_i and q_f denote the initial and end joint configurations for the base joint. q_r is the generated position reference of the base joint.

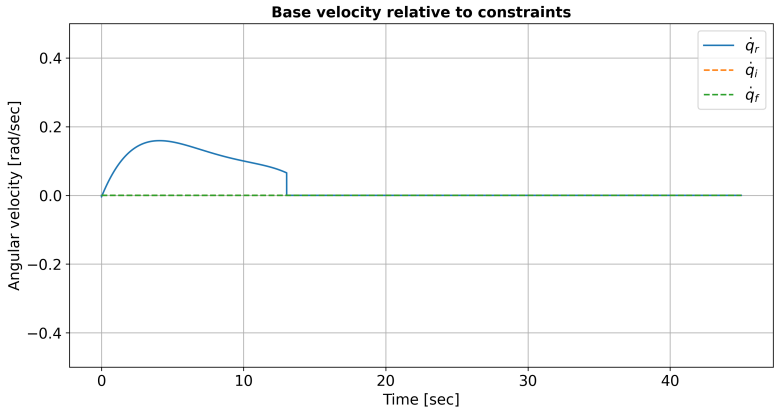


Figure 92: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for the base. $\dot{q}_r$ is the generated velocity reference of the base.

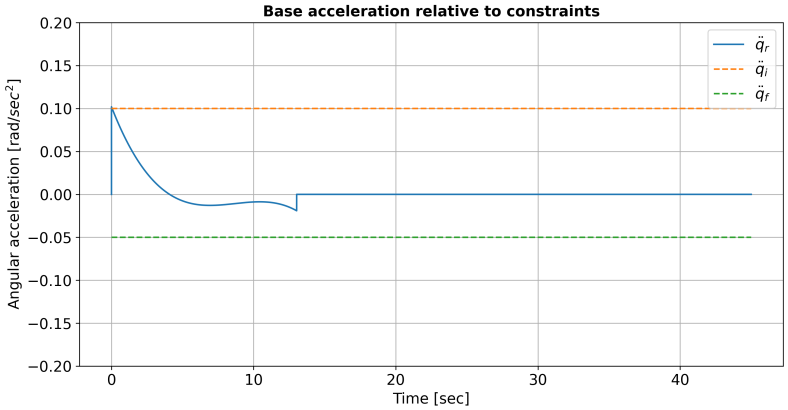


Figure 93: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for the base. $\ddot{q}_r$ is the generated acceleration reference of the base.

The figures Figure 91, Figure 92 and Figure 93 show the position, velocity and acceleration profile generated from point-to-point motion for the base of the crane. It can be seen that around 12 $[s]$ the position, velocity as well as the acceleration profile becomes zero. This is due to the system toggling from point-to-point motion and entering sway-compensation and station-keeping.

8.5.5 Point-to-point motion performance [joint1]

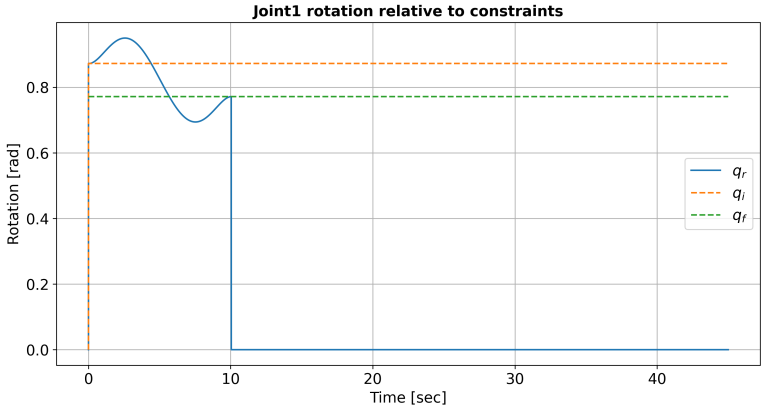


Figure 94: Here q_i and q_f denote the initial and end joint configurations for joint1. q_r is the generated position reference of joint1.

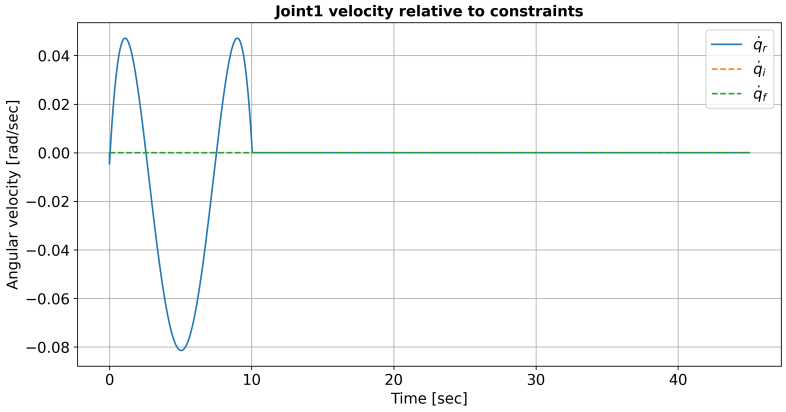


Figure 95: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for joint1. $\dot{q}_r$ is the generated velocity reference of joint1.

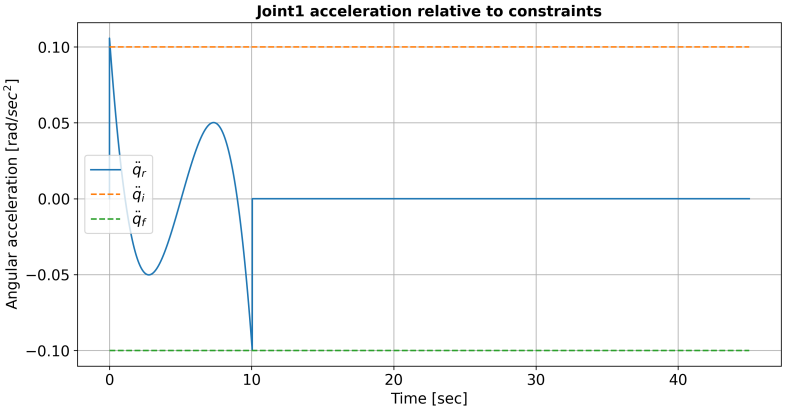


Figure 96: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for joint1. $\ddot{q}_r$ is the generated acceleration reference of joint1.

The figures shown in Figure 94, Figure 95 and Figure 96 show the position,velocity and acceleration profiles delivered from point-to-point motion regarding joint1. It can be seen that at 10 $[s]$ all profiles reach their desired end constraints, and are toggled to 0. The toggling occurs either when the joint has reached its desired position or the end-effector is sufficiently close to the end goal.

8.5.6 Point-to-point motion performance [joint2]

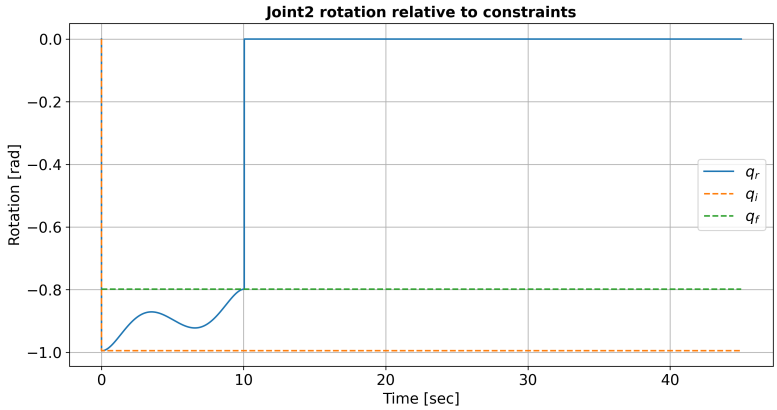


Figure 97: Here q_i and q_f denote the initial and end joint configurations for joint2. q_r is the generated position reference of joint2.

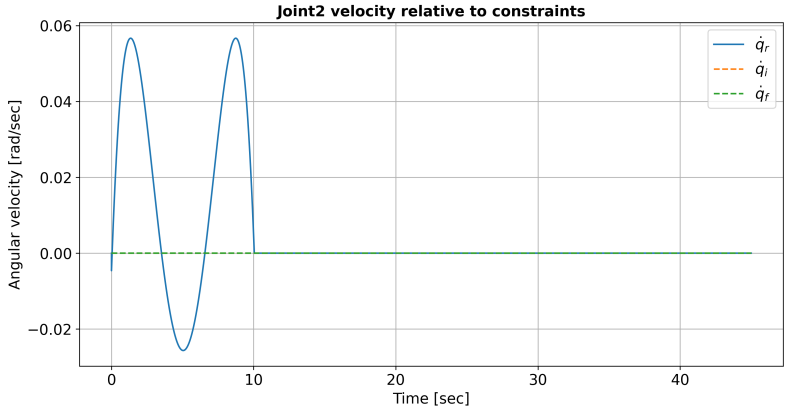


Figure 98: Here $\dot{q}_i$ and $\dot{q}_f$ denote the initial and end joint velocities for joint2. $\dot{q}_r$ is the generated velocity reference of joint2.

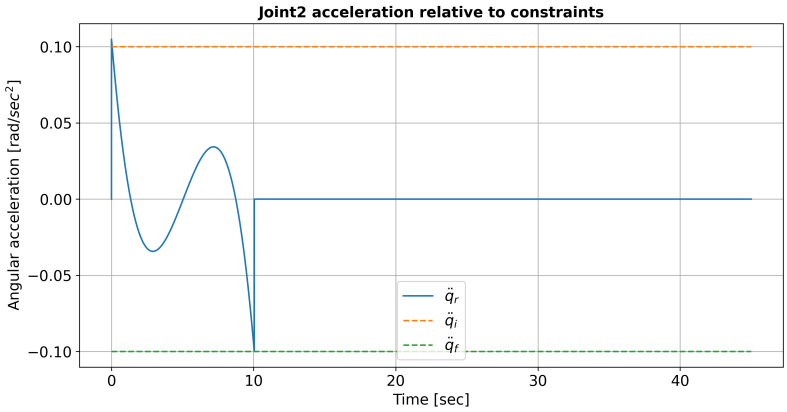


Figure 99: Here $\ddot{q}_i$ and $\ddot{q}_f$ denote the initial and end joint accelerations for joint2. $\ddot{q}_r$ is the generated acceleration reference of joint2.

Observing Figure 97, Figure 98 and Figure 99 they display similar behavior as seen in Section 8.5.5, where the position,velocity and acceleration profile perform as expected until 10 $[s]$, where the joint has reached its constraint and is set to 0.

8.5.7 Sway-compensation performance [x-direction]

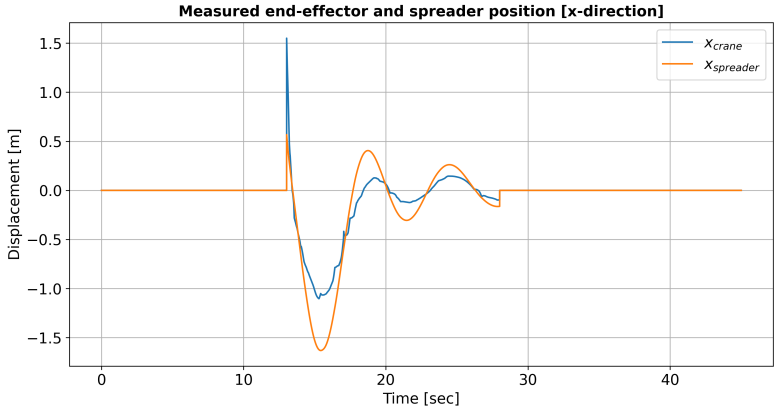


Figure 100: Here x_{crane} and $x_{spreader}$ are the x-positions of the crane-tip and the spreader mechanism

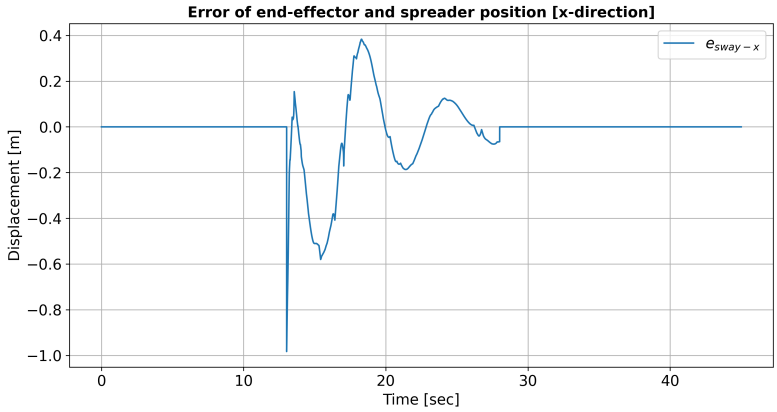


Figure 101: e_{sway-x} is the distance between the spreader and crane tip in x-direction.

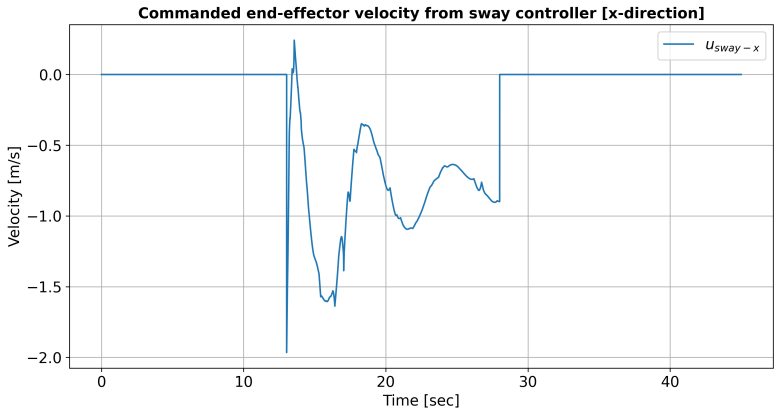


Figure 102: u_{sway-x} is the commanded end effector velocity from the sway controller in x direction.

The sway-compensation starts with a significant initial error. This is caused in part of the point-to-point motion algorithm, which generates a trajectory which induces pendulum motions on the spreader. Another factor is that the station-keeping controllers of the crane tip makes the crane-tip accelerate more than the point-to-point motion algorithm does, thus inducing pendulum motions. However the pendulum motions die out as can be seen in the error shown in Figure 101. Lastly the commanded end-effector velocity shown in Figure 102 shows the commended velocity stabilizing at approximately -1. This could be due to the integral gain term in the sway controller competing against the station-keeping controller. Similarly to Section 8.3.3, the crane-tip position is filled with spikes. This could again be an effect from the controllers experiencing small numerical errors due to the timestep of the solver, which when fed into the simplified actuators introduce errors into the control-loop, further amplifying the undesired behavior.

8.5.8 Sway-compensation performance [y-direction]

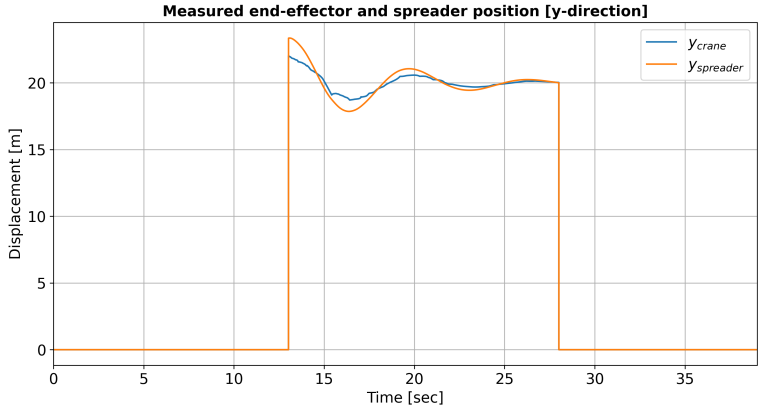


Figure 103: Here y_{crane} and $y_{spreader}$ are the y-positions of the crane-tip and the spreader mechanism

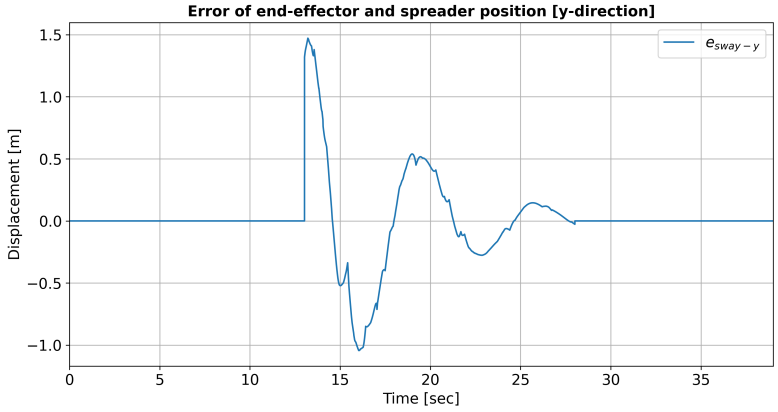


Figure 104: e_{sway-y} is the distance between the spreader and crane tip in y-direction.

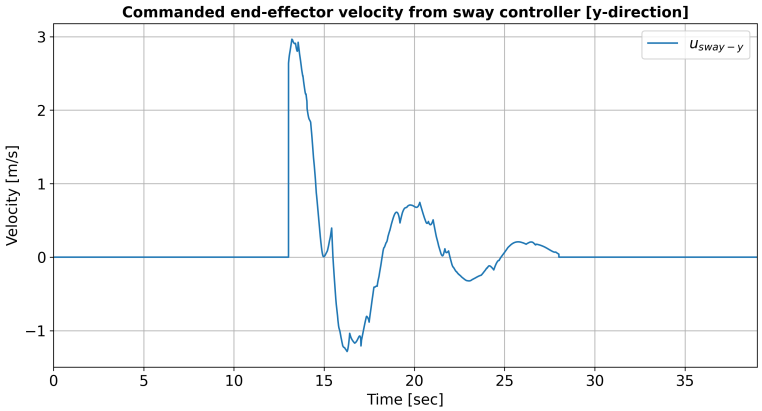


Figure 105: u_{sway-y} is the commanded end effector velocity from the sway controller in y direction.

The sway controller performance is similar to that seen in Section 8.5.7. The initial offset in crane and spreader position is significant at approximately 12 $[s]$ as shown in Figure 103. Additionally the error converges to 0 as shown in Figure 104. Lastly the commanded velocity input shown in Figure 105 converges to 0 $[m/s]$.

8.5.9 Station-keeping performance [x-direction]

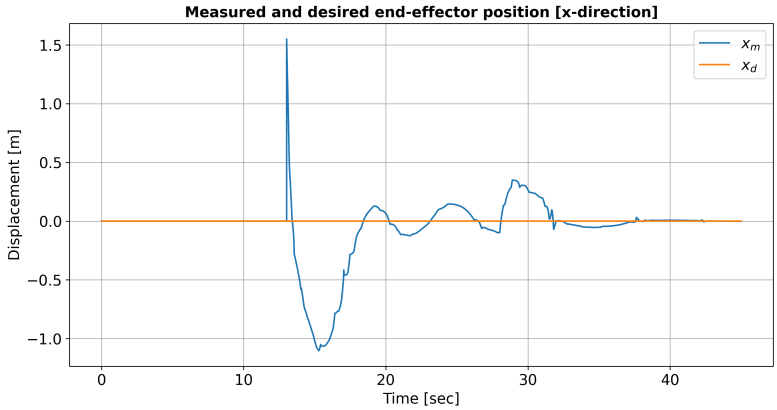


Figure 106: Here x_m and x_d is the measured and desired crane positions in x-direction.

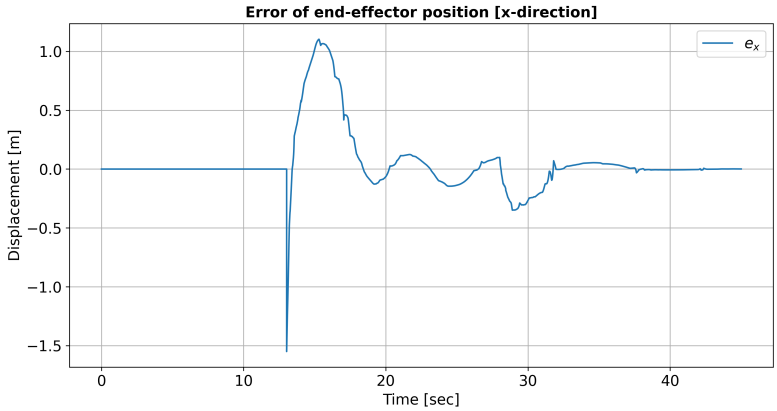


Figure 107: Here e_x is the error between the desired and measured crane tip position in x-direction.

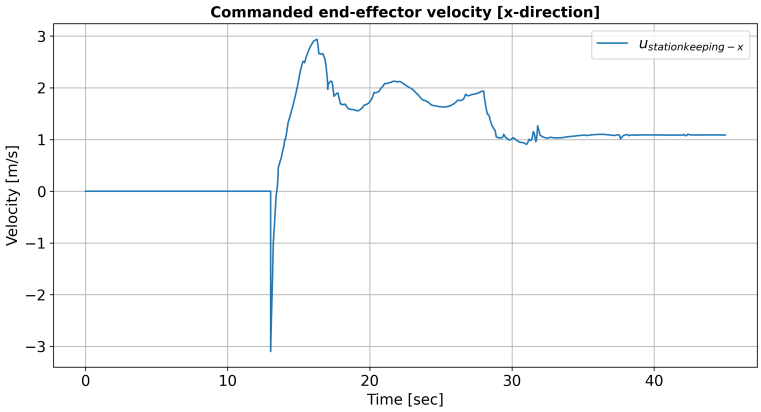


Figure 108: Here $u_{stationkeeping-x}$ is the commanded crane-tip velocity in x direction delivered from the station-keeping controller.

Investigating the measured and desired end-effector position shown in Figure 106, it can be seen that the controller experiences a large difference between the desired and measured crane tip position at approximately 12 $[s]$. At approximately 27 $[s]$ one can see that sway control gets toggled off in Figure 84. This corresponds to when the measured end-effector has a slight jump. The jump in position could be due to the sway controller no longer competes with the station-keeping controller. Lastly it can be seen that the error shown in Figure 107 goes to zero after approximately 33 $[s]$. The commanded control in Figure 108 seems to stabilize at 1. However this does not make sense as the sway-controller is turned off, and the error is 0.

8.5.10 Station-keeping performance [y-direction]

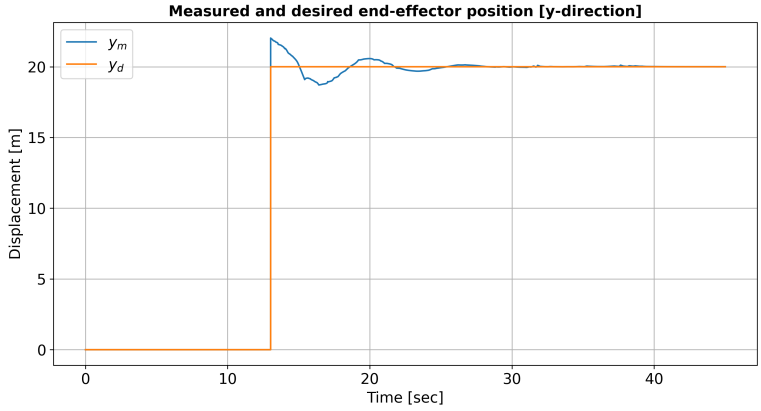


Figure 109: Here y_m and y_d is the measured and desired crane positions in y-direction.

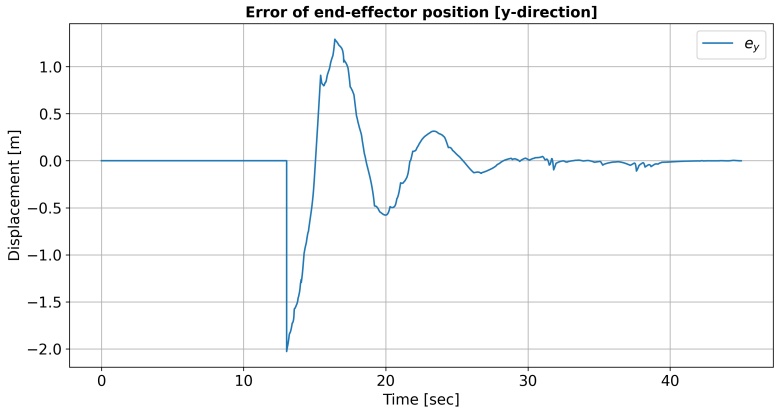


Figure 110: Here e_y is the error between the desired and measured crane tip position in y-direction.

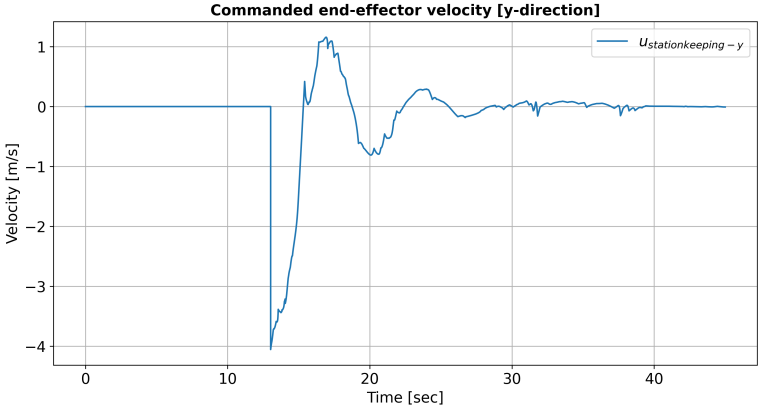


Figure 111: Here $u_{stationkeeping-y}$ is the commanded crane-tip velocity in y direction delivered from the station-keeping controller.

Inspecting Figure 109 showing the measured and desired crane position, it can be seen that the reference and measurement jumps from 0 to 20 $[m]$. This is due to the references and measurements of all station-keeping as well as sway-controllers are set to 0 while deactivated. Around 12 $[s]$, some minor oscillations of the end-effector position can be observed, and as the time increases the measurement converges towards to the desired position. Additionally Figure 110 which shows the error of the end-effector can be seen to oscillate in accordance to the results shown in Figure 109, before finally converging to 0 around 40 $[s]$. The commanded control input shown in Figure 111 performs as expected and converges to 0.

8.5.11 Station-keeping performance [z-direction]

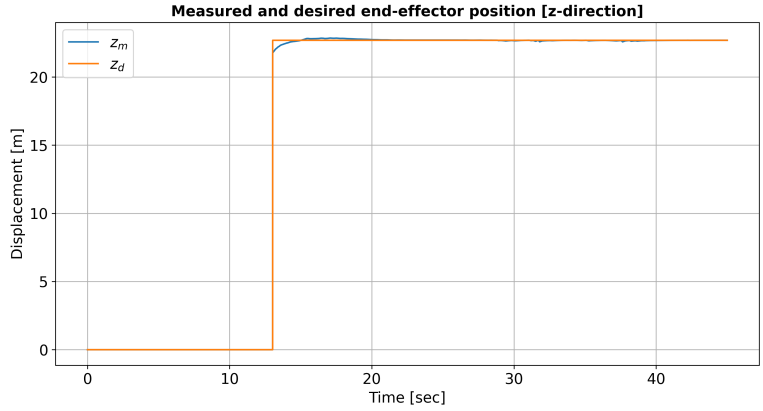


Figure 112: Here z_m and z_d is the measured and desired crane positions in z direction.

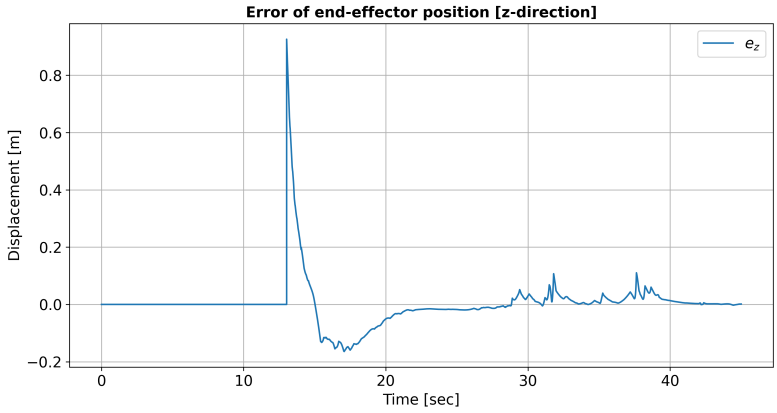


Figure 113: Here e_z is the error between the desired and measured crane tip position in z -direction.

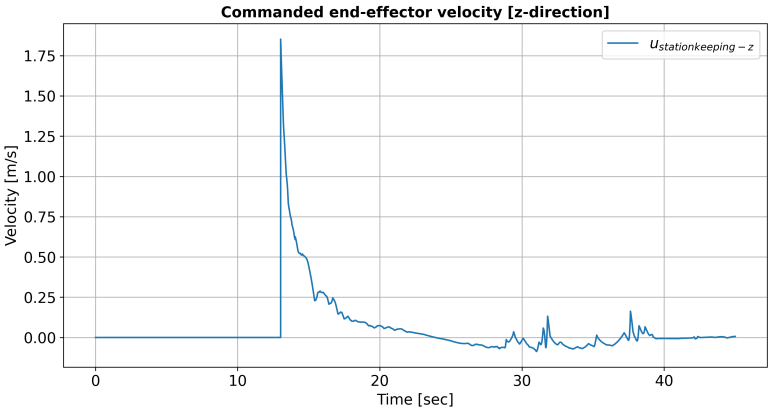


Figure 114: Here $u_{stationkeeping-z}$ is the commanded crane-tip velocity in z direction delivered from the station-keeping controller.

In Figure 112, one can see that around 12 $[s]$ the station-keeping controller in z-direction is activated, and the reference and measurement jumps from 0 to 17.5 $[m]$. It can also be seen that the reference and measurement converges around 20 $[s]$. This can also be seen in the error shown in Figure 113, where the error dies out after 20 $[s]$. Lastly the commanded control input shown in Figure 114 has an intial spike in control input, and then proceeds to stay around 0 after 20 $[s]$. In both Figure 113 and Figure 114 one can observe small spikes in both the error and control input. These should preferably be filtered out of the control loop. The possible reason for these are discussed in Section 8.5.7.

8.5.12 Wire lowering performance

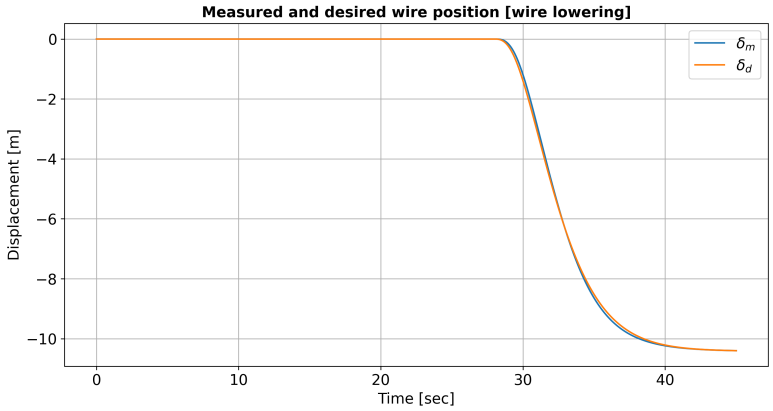


Figure 115: δ_m and δ_d is the measured and desired wire lengths relative to an initial wire length of 2.5 $[m]$.

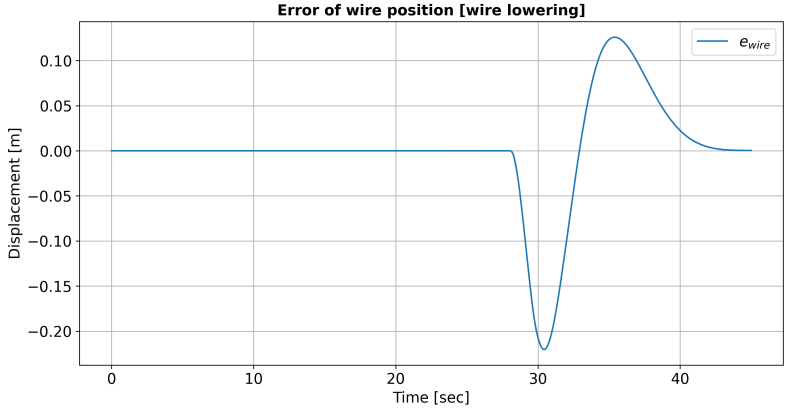


Figure 116: e_{wire} is the error between the measured and desired wire lengths.

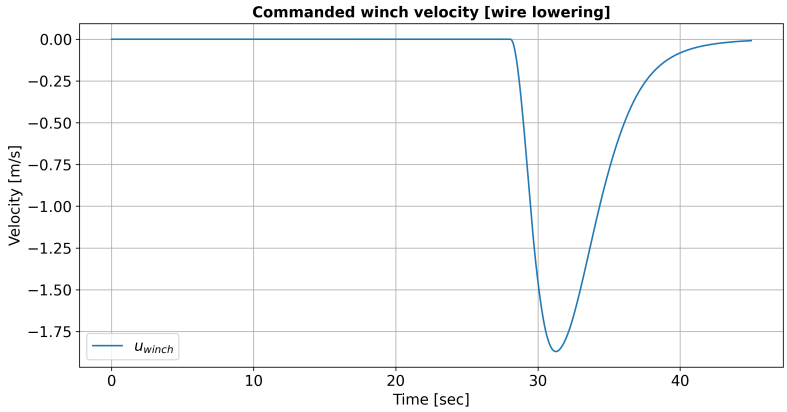


Figure 117: u_{winch} is the commanded winch velocity.

The measured and desired wire length relative to the initial wire of length 5 $[m]$ are shown in Figure 115. Here it can be seen that the reference filter starts generating a trajectory once the wire lowering is toggled in Figure 84. The controller is then able to follow the reference a relatively small error as can be seen in Figure 116, where the error is at most 0.21 $[m]$ from the reference. Lastly the commanded winch velocities shown in Figure 117 show that the commanded winch velocity converges to 0 after 40 $[s]$.

8.5.13 Revolt response

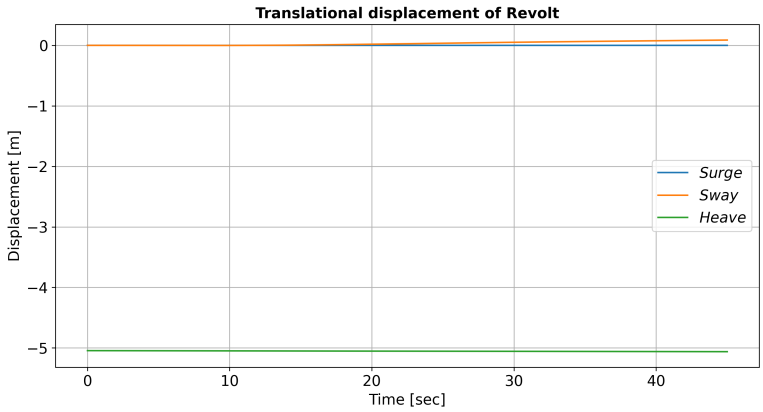


Figure 118: Position of Revolt in surge, sway and heave in terms of the inertial frame.

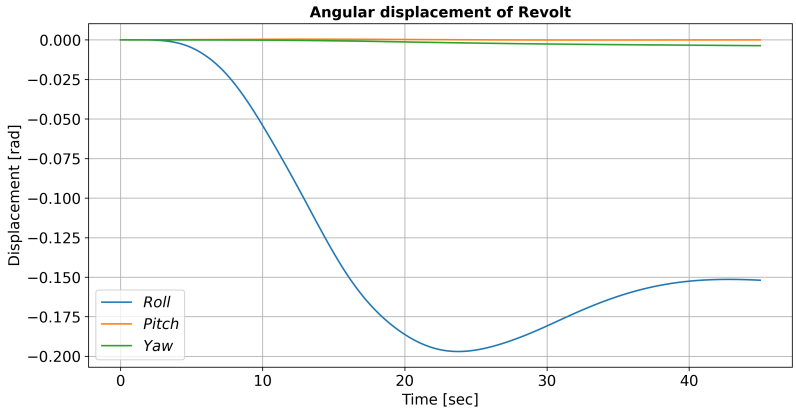


Figure 119: Angular displacements of Revolt in roll,pitch and yaw.

Inspecting Figure 118 which shows the traslational motions of Revolt in the inertial frame, it can be seen that in ideal conditions the position in heave measured from the keel of the ship is stable at what the draught is expected to be. Additionally the motions in surge and sway of the ship is quite small, where the ship sways slightly to the side as seen around 40 $[s]$. This could be due to the inertial force of the crane, as it extends from an an initial stern position to port-side. Figure 119 shows the angular displacements of the ship and it can be seen that the roll angle is quite large and stabilizes around 8.5°. Such roll angles during a loading operation under ideal conditions is unacceptable, and comes as a result of the increased vertical center of mass from the crane as seen in Section D. This should be rectified in further work by either scaling down the crane, or using ballast tanks, as this would lower the vertical center of gravity and then improve the stability characteristics of the complete system. Additionally it can also be seen that there is a small change in yaw angle around 40 $[s]$. This also makes sense as the inertial force from the crane should induce a yaw-motion on the ship.

9 Conclusion and further work

In order to accomplish a autonomous loading operation the thesis presents an integrated simulation model as well as a functional control system. The simulation model contains a ship, crane, wire and spreader, while the control system is able to correctly toggle between the tasks such as crane transport, sway compensation and wire-lowering.

Each of the control algorithms have been found to work correctly by themselves, but the integrated system show unwanted behavior. The rapid transition between point-to-point motion and sway compensation and station-keeping induces pendulum motions on the spreader, and the sway controllers seem to compete with the station-keeping controllers, which create a small unwanted offset until sway compensation gets turned off.

The point-to-point algorithm allows the crane to reach a desired end configuration, but should be replaced with a point-chaining algorithm, or be able to change its trajectory based on obstructions in the generated trajectory of the crane.

Additionally the integrated system is found to be stable in the worst case configurations, but the stability should be improved. This can be done by scaling down the crane, or adding weight in the bottom of Revolt in order to decrease the height of the vertical center of gravity.

The integrated system also lack any way of controlling the slewing motions of the spreader, which is an important part of the loading operation, and actuators that allow rotational control of the cargo such as tugger-wires should be implemented into the integrated model.

The workspace as well as all actuator models should also be improved. The current workspace of the crane does not take invalid crane configurations or collisions between crane links into account. All actuators have also been simplified to provide velocities directly and as such will be able to respond to any control input without delay or fail. In further work, the actuators of the crane should be modeled such that any actuator delay, weight and inertia is included in the integrated system.

Additionally the state-machine only takes the transport of the spreader to the container in the world into account, and in further work should be expanded upon to handle all phases of an loading operation as well as include fallback states should the system experience unexpected events and hazards. Lastly, the thesis has focused on developing the control system and a simulation model that is able to perform a loading operation. Due to the case considered, environmental models has not been the focus of the thesis, but in further work it could be of interest to implement environmental models in the form wind, waves and current in order to investigate system performance in conditions closer to real life. This would however demand station-keeping of the Revolt vessel to be implemented in the integrated model.

9.0.1 Further work summarized

- Make crane transportation algorithm take obstacles into account
- Make sway and station-keeping controllers not compete with each-other
- Include restrictions in crane workspace
- Model tugger wires to control rotations of spreader mechanism

- Improve stability properties of Revolt-crane system
- Include fallback states
- Create higher fidelity winch and crane joint actuators
- Include more states to FSM to automate more parts of a loading operation
- Include environmental models and include more cases for integrated system

Bibliography

[1] Contech. *INSTRUCTION FLYER SEMI-AUTOMATIC SPREADERS*. Accessed: 30 Nov, 2023. URL: https://www.containertechnics.com/blog/instruction-flyer-semi-automatic-spreaders.

[2] Mariann Merz et al. 'A gap analysis for automated cargo handling operations with geared vessels frequenting small sized ports'. In: *Maritime Transport Research* 5 (2023), p. 100098.

[3] Geraldine Knatz, Theo Notteboom and Anthanasios A. Pallis. 'Container terminal automation: revealing distinctive terminal characteristics and operating parameters'. In: *Maritime Economics Logistics* 24 (2022), pp. 537–565.

[4] Fredrik Gyberg. *Design, modeling and control of a generic crane for marine applications*. 2017.

[5] Henrik Hammarstrøm. *Simulated Autonomous Loading Operations*. 2023.

[6] Børge Rokseth, Stian Skjong and Eilif Pedersen. 'Modeling of Generic Offshore Vessel in Crane Operations With Focus on Strong Rigid Body Connections'. In: *IEEE Journal of Oceanic Engineering* 42.4 (2017), pp. 846–868.

[7] Dean C. Karnopp, Donald L. Magrolis and Ronald C. Rosenberg. *System Dynamics: Modeling, Simulation, and Control of Mechantronic Systems*. John Wiley and Sons, Inc., 2012.

[8] Hallvard Engja Eilif Pedersen. *Mathematical modelling and simulation of physical systems : lecture notes in course TMR4275 modelling, simulation and analysis of dynamic systems*. eng. Trondheim, 2014.

[9] Leonard Meirovitch. *Methods of analytical dynamics*. eng. New York, 1970.

[10] Lorenzo Sciavicco. *Modelling and Control of Robot Manipulators*. eng. London, 2000.

[11] Krishnakumar N, Savitha Ramasamy and Abdullah Al-Mamun. *Active vibration compensator on moving vessel by hydraulic parallel mechanism - Scientific Figure on ResearchGate*. Accessed: 18 Nov, 2023. 2018. URL: https://www.researchgate.net/figure/Six-degrees-of-freedom-for-ship-motion_fig1_327901742.

[12] *Marin teknikk grunnlag : TMR4105 : kompendium*. nob. Trondheim, 2015.

[13] Ronald C Arkin. *Behavior-based robotics*. eng. Cambridge, Mass, 1998.

[14] Mae L. Seto. *Marine Robot Autonomy*. eng. Vol. 9781461456599. 2013. ISBN: 1461456592.

[15] R. Brooks. 'A robust layered control system for a mobile robot'. In: *IEEE Journal on Robotics and Automation* 2.1 (1986), pp. 14–23.

[16] Mordechai Ben-Ari and Francesco Mondada. 'Finite State Machines'. In: Jan. 2018, pp. 55–61.

[17] Thor I. Fossen. *Handbook of Marine Craft Hydrodynamics and Motion Control*. Wiley, 2011.

[18] O. M. Faltinsen. *Sea Loadds on Ships and Offshore Structures*. Cambridge University Press, 1990.

[19] Jens G Balchen. *Reguleringsteknikk*. nob. Trondheim, 2016.

[20] Yingguang Chu et al. 'An Effective Heave Compensation and Anti-sway Control Approach for Offshore Hydraulic Crane Operations'. In: Aug. 2014.

[21] Kjell Larsen. *TMR4225 Marine Operations. Lecture notes - Subsea Lifting Operations - part2 of 2*. Jan. 2022.

[22] Asgeir J. Sørensen. *Marine Cybernetics, Towards Autonomous Marine Operations and Systems, Lecture Notes*. Department of Marine Technology, NTNU, 2018.

[23] Palfinger. *Knuckle Boom Cranes*. Accessed: 04. June, 2024. URL: https://www.palfingermarine.com/en/deck-equipment/cranes/knuckle-boom-cranes#abd64e4eccollapse2.

[24] Controllab. *20-Sim 5.0 Reference Manual*. Accessed: 7 June, 2024. URL: https://www.20sim.com/help/manuals/.

[25] OpenAI. *ChatGPT (23. mars version)*. Large Language Modell. 2023. URL: https://chat.openai.com/.

Appendix

A Ship-crane integration

Integrating ship and crane

Conventions:

- R{b}_{a} represents the transform from frame **{a}** to frame **{b}.**
- r_{a}_{b}_{c} represents the vector going from frame **{c}** to frame **{b}** described in terms of frame **{a}**.

Defenitions of generalized and quasi coordinates:

The generalized coordinates of the total system is: **q = [(r_b/0^0)^T , Theta_b/0**^0^**T, q_e^T]^T**.

- Here **(r_b/0^0)**^T is the position vector from inertial frame {0} to the body fixed frame of the ship{b} expressed in terms of the inertial frame {0}.
- Theta^T is the vector denoting the rotations in x, y and z axis of the body-fixed frame {b} expressed in terms of the inertial frame {0}
- **q_e**^T = [theta_1, theta_2, theta_3]^T , which represents the rotations of each of the crane joints from joint 1-3.

The quasi coordinates of the system is defined as the time rates of the movement and rotations of the vessel in body frame, and the time rates of the generailzed coordinates representing the rotations of teh crane joints.

omega = [v_b/0^b, omega_b/0^b, q_dot_ e]^T

NB!

Body frame is defined as half the length of the perpendiculars on deck. Location of body fixed frame of vessel relative to bottom left on deck: (x = Lpp/2, y = 0, z = H)

Defining elementary rotation matrixes:

```
restart:
with(LinearAlgebra):
with(VectorCalculus):
with(CodeGeneration): with(codegen, optimize, makeproc):
```

Rotation matrix relating from body {b} to inerial frame {0}

```
Rx := Matrix([[1, 0, 0], [0, cos(ϕ), -sin(ϕ)], [0, sin(ϕ), cos(ϕ)]]):
Ry := Matrix([[cos(θ), 0, sin(θ)], [0, 1, 0], [-sin(θ), 0, cos(θ)]]):
Rz := Matrix([[cos(ψ), -sin(ψ), 0], [sin(ψ), cos(ψ), 0], [0, 0, 1]]):
Rb_0 := simplify(Rz Ry Rx):

Rx_inv := Transpose(Rx):
```

$Ry\_inv := Transpose(Ry)$:
$Rz\_inv := Transpose(Rz)$:

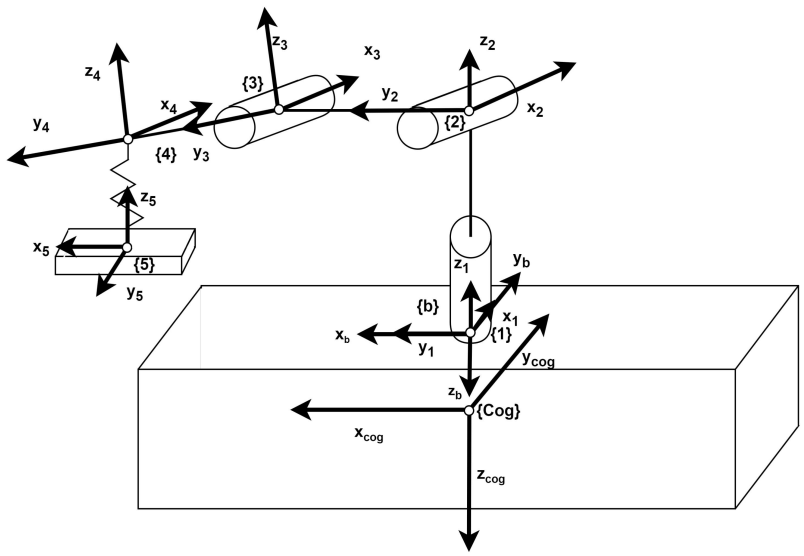
Defining quasi coordinates transforms

$i\_b := Vector([1, 0, 0])$:
$j\_b := Vector([0, 1, 0])$:
$k\_b := Vector([0, 0, 1])$:

$T\_inv := Matrix([i\_b, Rx\_inv\ j\_b,\ Rx\_inv\ Ry\_inv\ k\_b])$:

$alpha\_T := Matrix([[Transpose(Rb\_0),\ ZeroMatrix(3, 3),\ ZeroMatrix(3, 3)],$
$[ZeroMatrix(3, 3), T\_inv, ZeroMatrix(3, 3)],$
$[ZeroMatrix(3, 3), ZeroMatrix(3, 3), IdentityMatrix(3)]])$:
beta $:= MatrixInverse(alpha\_T)$:
alpha $:= Transpose(alpha\_T)$:

Defining crane parameters and transforms:



GYBERG PARAMETERS

Defining delta1, as a function of theta2.

$u\_x := \sin(theta2) \cdot u$:
$b\_x := \cos(theta2) \cdot b$:
$e\_x := b\_x + u\_x - a$:
$b\_z := \sin(theta2) \cdot b$:

$u\_z := \cos(theta2)\cdot u$:
$e\_z := h + b\_z - u\_z$:
$e := \mathrm{sqrt}\left(e\_z^2 + e\_x^2\right)$:
$delta1 := \arcsin\left(\frac{e\_z}{e}\right)$:

Warning, if e is meant to be the exponential e, use command/symbol completion or palettes to enter this special symbol, or use the exp function

Defining delta2, as a function of theta2, and theta3.

$c\_mark := \mathrm{sqrt}\left(s^2 + c^2\right)$:
$beta2 := \arcsin\left(\frac{s}{c\_mark}\right)$:
$my2 := \mathrm{pi} + theta3 - beta2 - epsilon2$:
$v := \mathrm{sqrt}\left(r^2 + (L1 - w)^2\right)$:
$epsilon2 := \arcsin\left(\frac{r}{v}\right)$:
$d := \mathrm{sqrt}\left(v^2 + c\_mark^2 - 2\cdot(v\cdot c\_mark\cdot\cos(my2))\right)$:
$phi2 := solve\left(v = \mathrm{sqrt}\left(d^2 + c\_mark^2 - 2\cdot(d\cdot c\_mark\cdot\cos(Phi2))\right), Phi2\right)$:
$alpha2 := \mathrm{pi} - \left(\frac{\pi}{2} - theta2\right) + theta3 - beta2$:
$kappa2 := \mathrm{pi} - alpha2 - phi2$:
$delta2 := \frac{\mathrm{pi}}{2} - kappa2$:

Defining all rotation matrixes:

#from {1} to {b}
$Rb\_1 := Matrix([[\cos(\mathrm{Pi}), 0, \sin(\mathrm{Pi})], [0, 1, 0], [-\sin(\mathrm{Pi}), 0, \cos(\mathrm{Pi})]])$ $Matrix\left(\left[\left[\cos\left(\frac{\mathrm{Pi}}{2}\right), -\sin\left(\frac{\mathrm{Pi}}{2}\right), 0\right], \left[\sin\left(\frac{\mathrm{Pi}}{2}\right), \cos\left(\frac{\mathrm{Pi}}{2}\right), 0\right], [0, 0, 1]\right]\right)$ $Matrix([[\cos(theta1), -\sin(theta1), 0], [\sin(theta1), \cos(theta1), 0], [0, 0, 1]])$:
$R1\_b := Rb\_1^{-1}$:

#from {2} to {1}
$R1\_2 := Matrix([[1, 0, 0], [0, \cos(theta2), -\sin(theta2)], [0, \sin(theta2), \cos(theta2)]])$:

 # from {3} to {2}
$R2\_3 := Matrix([[1, 0, 0], [0, \cos(theta3), -\sin(theta3)], [0, \sin(theta3), \cos(theta3)]])$:

Local corrdinate frame positions

```
rb_1_b := Vector([0, 0, 0]) : #position of frame 1 relative to frame b
r1_2_1 := Vector([0, 0, h]) : #position of frame 2 relative to frame 1
r2_3_2 := Vector([0, L1, 0]) : #position of frame 3 relative to frame 2
r3_4_3 := Vector([0, L2, 0]) : #position of frame 4 relative to frame 3

r1_cm2_1 := Vector([Xcm1, Ycm1, 0]) : #centre of gravity of base realtive to frame 1
r2_cm3_2 := Vector([0, k, 0]) : #centre of gravity of  lower arm (link1) realtive to frame 2
r3_cm4_3 := Vector([0, n, 0]) : #centre of gravity of upper arm (link2) realtive to frame 3
```

Positions in terms of the b-frame

```
rb_2_1 := rb_1_b + Rb_1 r1_2_1 :
rb_3_2 := Rb_1 R1_2 r2_3_2 :
rb_4_3 := Rb_1 R1_2 R2_3 r3_4_3 :

rb_2_b := rb_1_b + rb_2_1 :
rb_3_b := rb_2_b + rb_3_2 :
rb_4_b := rb_3_b + rb_4_3 :
```

Unit vectors that each joint revolve about

```
i_vect := Vector([1, 0, 0]) :
 j_vect := Vector([0, 1, 0]) :
k_vect := Vector([0, 0, 1]) :

e1 := Rb_1 k_vect : #rotation of base
e2 := Rb_1 R1_2 i_vect : #rotation of lower arm
e3 := Rb_1 R1_2 R2_3 i_vect : #rotation of upper arm
```

Position for each Center of mass defined relative to and in terms of base frame

```
rb_cm2 := Rb_1 r1_cm2_1 :#Mass of the platform
rb_cm3 := rb_2_b + Rb_1 R1_2 r2_cm3_2 :#Mass of lower arm
rb_cm4 := rb_3_b + Rb_1 R1_2 R2_3 r3_cm4_3 : #Mass of upper arm
```

Defining geometric jacobian of vessel:

```
r_cg1_0_b := Vector([X_cg, Y_cg, Z_cg]) :
j_cg1_0_vb := Matrix([IdentityMatrix(3), CrossProduct(i_b, r_cg1_0_b), CrossProduct(j_b,
     r_cg1_0_b), CrossProduct(k_b, r_cg1_0_b)]) :
 j_cg1_0_wb := Matrix([ZeroMatrix(3, 3), IdentityMatrix(3), ]) :
Jcm1b := Matrix([[j_cg1_0_vb, ZeroMatrix(3, 3)], [j_cg1_0_wb, ZeroMatrix(3, 3)]]) :
     # Geometric jacobian relating quasi coordinates to vessel velocity
```

Geometric Jacobian for each center of mass

```
# Finding geometric jacobian relating quasi coordinates to base centre of gravity
rb_cm2_1 := rb_cm2 − rb_1_b :
 jcm2v_0_vb := Matrix( [IdentityMatrix(3) ]) :
jcm2v_0_wb := Matrix( [CrossProduct(i_b, rb_cm2_1), CrossProduct( j_b, rb_cm2_1),
      CrossProduct(k_b, rb_cm2_1) ]) :
 jcm2v_0_qe := Matrix( [CrossProduct(e1, rb_cm2_1), ZeroMatrix(3, 2) ]) :
Jcm2v := Matrix( [ [jcm2v_0_vb, jcm2v_0_wb,  jcm2v_0_qe]]) :

 jcm2w_0_vb := Matrix( [ZeroMatrix(3, 3) ]) :
 jcm2w_0_wb := Matrix( [IdentityMatrix(3, 3) ]) :
 jcm2w_0_qe := Matrix( [e1, ZeroMatrix(3, 2) ]) :
Jcm2w := Matrix( [ jcm2w_0_vb, jcm2w_0_wb , jcm2w_0_qe ]) :
Jcm2b := Matrix( [ [Jcm2v], [Jcm2w]]) :

  # Finding geometric jacobian relating quasi coordinates to centre of  gravity of lower arm (link1)
rb_cm3_1 := rb_cm3 − rb_1_b :
rb_cm3_2 := rb_cm3 − rb_2_b :

 jcm3v_0_vb := Matrix( [IdentityMatrix(3) ]) :
jcm3v_0_wb := Matrix( [CrossProduct(i_b,  rb_cm3), CrossProduct( j_b,  rb_cm3) ,
      CrossProduct(k_b,  rb_cm3) ]) :
 jcm3v_0_q1 := Vector(CrossProduct(e1,  rb_cm3_1) ) :
 jcm3v_0_q2 := Vector(CrossProduct(e2, rb_cm3_2) ) :
 jcm3v_0_q3 := Vector( [0, 0, 0 ]) :
Jcm3v := Matrix( [ jcm3v_0_vb, jcm3v_0_wb, jcm3v_0_q1, jcm3v_0_q2, jcm3v_0_q3 ]) :

 jcm3w_0_vb := Matrix( [ZeroMatrix(3) ]) :
 jcm3w_0_wb := Matrix( [IdentityMatrix(3) ]) :
 jcm3w_0_q1 := e1 :
 jcm3w_0_q2 := e2 :
 jcm3w_0_q3 := Vector( [0, 0, 0 ]) :
Jcm3w := Matrix( [ jcm3w_0_vb,  jcm3w_0_wb,  jcm3w_0_q1, jcm3w_0_q2, jcm3w_0_q3 ]) :
Jcm3b := Matrix( [ [Jcm3v], [Jcm3w]]) :

  # Finding geometric jacobian relating quasi coordinates to centre of  gravity of  upper arm (link2)
rb_cm4_1 := rb_cm4 − rb_1_b :
rb_cm4_2 := rb_cm4 − rb_2_b :
rb_cm4_3 := rb_cm4 − rb_3_b :

 jcm4v_0_vb := Matrix( [IdentityMatrix(3) ]) :
jcm4v_0_wb := Matrix( [CrossProduct(i_b,  rb_cm4), CrossProduct( j_b,  rb_cm4) ,
      CrossProduct(k_b,  rb_cm4) ]) :
 jcm4v_0_q1 := Vector(CrossProduct(e1, rb_cm4_1) ) :
 jcm4v_0_q2 := Vector(CrossProduct(e2, rb_cm4_2) ) :
 jcm4v_0_q3 := Vector(CrossProduct(e3, rb_cm4_3) ) :
Jcm4v := Matrix( [ jcm4v_0_vb,  jcm4v_0_wb, jcm4v_0_q1, jcm4v_0_q2, jcm4v_0_q3 ]) :

 jcm4w_0_vb := Matrix( [ZeroMatrix(3) ]) :
```

$jcm4w\_0\_wb := Matrix([IdentityMatrix(3)]):$
$jcm4w\_0\_q1 := e1:$
$jcm4w\_0\_q2 := e2:$
$jcm4w\_0\_q3 := e3:$
$Jcm4w := Matrix([jcm4w\_0\_vb,\ jcm4w\_0\_wb, jcm4w\_0\_q1, jcm4w\_0\_q2, jcm4w\_0\_q3]):$
$Jcm4b := Matrix([[Jcm4v], [Jcm4w]]):$

Mass inertia matrixes

#Ship
$I\_g\_ship := Matrix([[Ixx\_ship,\ -Ixy\_ship,\ -Ixz\_ship], [-Iyx\_ship, Iyy\_ship,\ -Iyz\_ship], [-Izx\_ship, -Izy\_ship, Izz\_ship]]):$

$M\_ship := Matrix(m\_ship \cdot IdentityMatrix(3)):$
$B\_ship := Transpose(Jcm1b)\ Matrix([[M\_ship\ , ZeroMatrix(3, 3)], [ZeroMatrix(3, 3), I\_g\_ship,]])\ Jcm1b:$

#Base
$I\_1\_1\_base := Matrix([[Ixx\_base,\ 0, 0], [0\ Iyy\_base, 0], [0, 0\ , Izz\_base]]):$
$I\_1\_b\_base := R1\_b\ I\_1\_1\_base\ Rb\_1:$
$M\_base := Matrix([[m\_base \cdot IdentityMatrix(3), ZeroMatrix(3, 3)], [ZeroMatrix(3), I\_1\_b\_base]]):$
$B2 := Transpose(Jcm2b)\ M\_base\ Jcm2b:$

#Lower arm (link1)
$I\_2\_2\_link1 := Matrix([[Ixx\_link1,\ 0, 0], [0, Iyy\_link1,\ 0], [0, 0, Izz\_link1]]):$
$I\_2\_b\_link1 := R2\_b\ I\_2\_2\_link1\ Rb\_2:$
$M\_link1 := Matrix([[m\_link1 \cdot IdentityMatrix(3), ZeroMatrix(3, 3)], [ZeroMatrix(3), I\_2\_b\_link1]]):$

$B3 := Transpose(Jcm3b)\ M\_link1\ Jcm3b:$

#Upper arm (link2)
$I\_3\_3\_link2 := Matrix([[Ixx\_link2,\ 0, 0], [0, Iyy\_link2, 0], [0, 0, Izz\_link2]]):$
$I\_3\_b\_link2 := R3\_b\ I\_3\_3\_link2\ Rb\_3:$
$M\_link2 := Matrix([[m\_link2 \cdot IdentityMatrix(3), ZeroMatrix(3, 3)], [ZeroMatrix(3), I\_3\_b\_link2]]):$

$B4 := Transpose(Jcm4b)\ M\_link2\ Jcm4b:$

$B\_tot := B\_ship + B2 + B3 + B4:$

Jacobian for crane-tip

#Crane Tip
$rb\_4\_1 := rb\_4\_b - rb\_1\_b:$
$rb\_4\_2 := rb\_4\_b - rb\_2\_b:$

$rb\_4\_3 := rb\_4\_b - rb\_3\_b$:

$j11v\_0\_vb := IdentityMatrix(3)$:
$j11v\_0\_wb := Matrix([CrossProduct(i\_b, rb\_4\_b), CrossProduct(j\_b, rb\_4\_b), CrossProduct(k\_b, rb\_4\_b)])$:
$j11v\_0\_q1 := CrossProduct(e1, rb\_4\_1)$:
$j11v\_0\_q2 := CrossProduct(e2, rb\_4\_2)$:
$j11v\_0\_q3 := CrossProduct(e2, rb\_4\_3)$:
$J11v := Matrix([j11v\_0\_vb, j11v\_0\_wb, j11v\_0\_q1, j11v\_0\_q2, j11v\_0\_q3])$:

$j11w\_0\_vb := ZeroMatrix(3, 3)$:
$j11w\_0\_wb := IdentityMatrix(3)$:
$j11w\_0\_q1 := e1$:
$j11w\_0\_q2 := e2$:
$j11w\_0\_q3 := e3$:
$J11w := Matrix([j11w\_0\_vb, j11w\_0\_wb, j11w\_0\_q1, j11w\_0\_q2, j11w\_0\_q3])$:

$J11b := Matrix([[J11v], [J11w]])$:

Differentiating alpha

$qv := Vector([\phi, \theta, \psi])$:
#We only inclue the euler angeles as taking the partial derivative of the other states gives 0

$dalphaij\_dq := Matrix(3, 81)$:

$col1 := Vector([1, 10, 19, 28, 37, 46, 55, 64, 73])$:
$col2 := Vector([2, 11, 20, 29, 38, 47, 56, 65, 74])$:
$col3 := Vector([3, 12, 21, 30, 39, 48, 57, 66, 75])$:
$col4 := Vector([4, 13, 22, 31, 40, 49, 58, 67, 76])$:
$col5 := Vector([5, 14, 23, 32, 41, 50, 59, 68, 77])$:
$col6 := Vector([6, 15, 24, 33, 42, 51, 60, 69, 78])$:
$col7 := Vector([7, 16, 25, 34, 43, 52, 61, 70, 79])$:
$col8 := Vector([8, 17, 26, 35, 44, 53, 62, 71, 80])$:
$col9 := Vector([9, 18, 27, 36, 45, 54, 63, 72, 81])$:

$iterator := Matrix([col1, col2, col3, col4, col5, col6, col7, col8, col9])$:

for i **from** 1 **to** 3 **do for** j **from** 1 **to** 9 **do for** k **from** 1 **to** 3 **do** $dalphaij\_dq[k, iterator[i, j]] := diff(\text{alpha}[i, j], qv[k])$:**end do end do end do**

Differentiating B_tot

$dBdq7 := simplify(map(diff, B\_tot, theta1))$:

$dBdq8 := simplify(\ map(diff, B\_tot, theta2\ )\ ):$
$dBdq9 := simplify(\ map(diff, B\_tot, theta3\ )\ ):$

Code export
Export the mass-inertia matrix, partial derivatives of B, and partial derivatives of alpha

$\#for\ i\ from\ 1\ to\ 9\ do\ for\ j\ from\ i\ to\ 9\ do\ \ F := makeproc(B\_tot[i,j]);\ C(F, resultname = cat("B", i, j),$
output = "exportedCode.txt", optimize) :**end do end do**:

$\#for\ i\ from\ 1\ to\ 9\ do\ for\ j\ from\ i\ to\ 9\ do\ \ F := makeproc(dBdq7[i,j]);\ C(F, resultname$
$= cat("dBdq7\_", i, j),$ output = "exportedCode.txt", optimize) :**end do end do**:

$\#for\ i\ from\ 1\ to\ 9\ do\ for\ j\ from\ i\ to\ 9\ do\ \ F := makeproc(dBdq8[i,j]);\ C(F, resultname$
$= cat("dBdq8\_", i, j),$ output = "exportedCode.txt", optimize) :**end do end do**:

$\#for\ i\ from\ 1\ to\ 9\ do\ for\ j\ from\ i\ to\ 9\ do\ \ F := makeproc(dBdq9[i,j]);\ C(F, resultname$
$= cat("dBdq9\_", i, j),$ output = "exportedCode.txt", optimize) :**end do end do**:

$\#for\ i\ from\ 1\ to\ 3\ do\ for\ j\ from\ 1\ to\ 81\ do\ \ F := makeproc(dalphaij\_dq[i,j]);\ C(F, resultname$
$= cat("dalphaij\_dq\_", i, j),$ output = "exportedCode.txt", optimize) :**end do end do**:

B Inertia calculations of Revolt

Revolt autocad inventor inertia values

Autocad inventor computes the inertia of revolt using the material density 0,298 g/cm^3. This is found from using the volume computed in Autocad, and inserting the mass of the fullscale Revolt to be 2 400 000 kg into the iProperties sheet in Autocad. The following parameters was found:

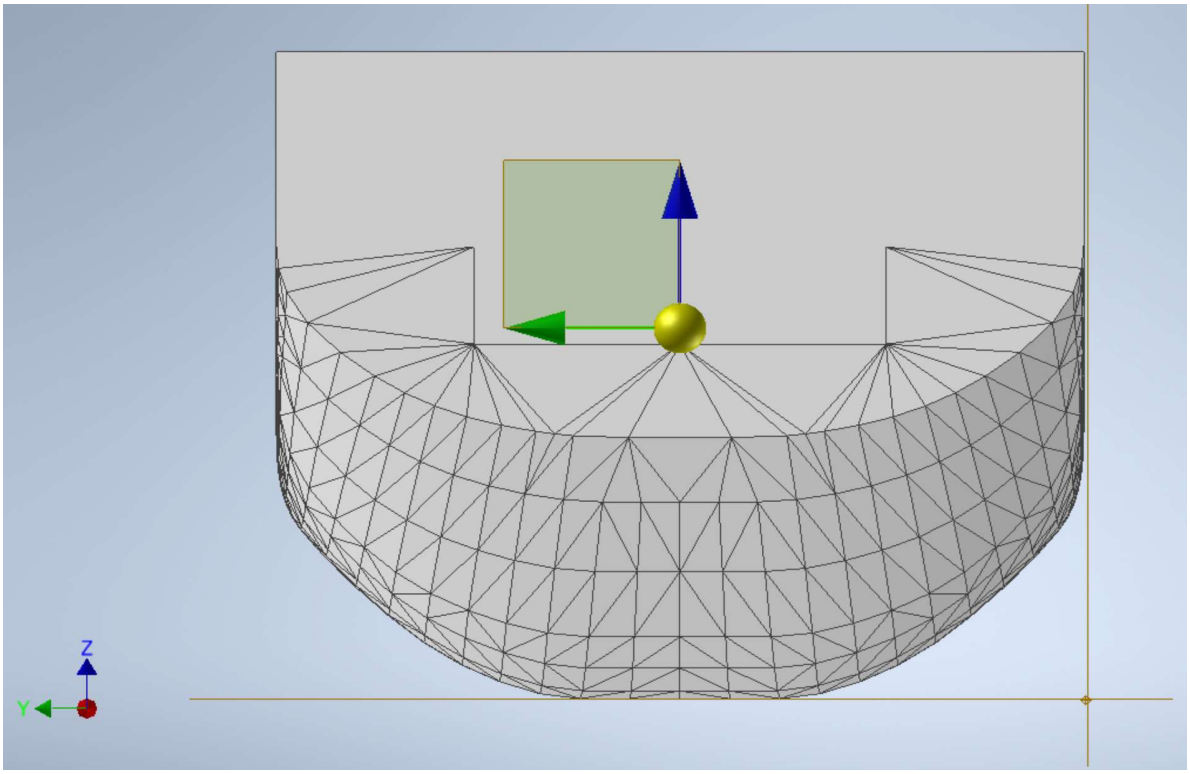
***Center of Gravity:*

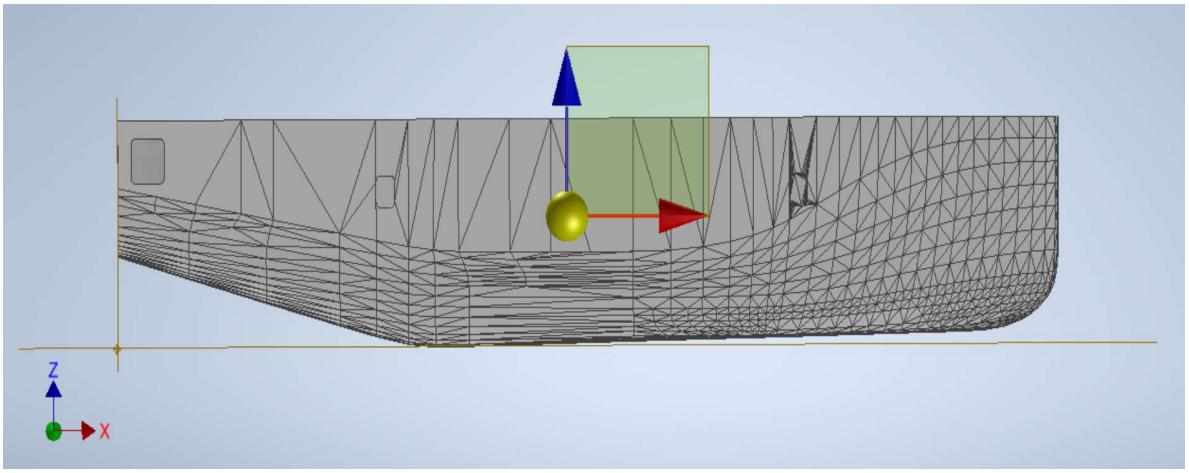
X: *28763,646 mm (Relative Error = 0,307593%)*
Y: *7318,761 mm (Relative Error = 0,307593%)*
Z: *6986,913 mm (Relative Error = 0,307593%)*

**Mass Moments of Inertia with respect to Center of Gravity(Calculated using negative integral)

Ixx 5,9107472699289195E+13 kg mm^2 (Relative Error = 0, 307593%)
Iyx 1852782120,100 kg mm^2 (Relative Error = 0,307593%)
Iyy 5,86959713478641250E+14 kg mm^2 (Relative Error = 0,307593%)
Izx 9,620599561592400E+12 kg mm^2 (Relative Error = 0, 307593%)
Izy -527698338,200 kg mm^2 (Relative Error = 0,307593%)
Izz 5,98623282290516000E+14 kg mm^2 (Relative Error = 0, 307593%)

The centre of gravity is found realtive to the bottom right of the ship shown below:





Converting from [mm] to [m]

Centre of gravity:

$X\_c := 28763.646 \cdot 10^{-3}$

$$X\_c := 28.76364600 \quad \textbf{(1)}$$

$$Y\_c := 7318.761 \cdot 10^{-3}$$

$$Y\_c := 7.318761000 \quad \textbf{(2)}$$

$$Z\_c := 6986.913 \cdot 10^{-3}$$

$$Z\_c := 6.986913000 \quad \textbf{(3)}$$

Inertias in [kg * m^2]:

$$Ixx := 5.9107472699289195E13 \cdot 10^{-6}$$

$$Ixx := 5.910747270 \times 10^{7} \quad \textbf{(4)}$$

$$Iyx := 1852782120.100 \cdot 10^{-6}$$

$$Iyx := 1852.782120 \quad \textbf{(5)}$$

$$Iyy := 5.86959713478641250E14 \cdot 10^{-6}$$

$$Iyy := 5.869597135 \times 10^{8} \quad \textbf{(6)}$$

$$Izx := 9.620599561592400E12 \cdot 10^{-6}$$

$$Izx := 9.620599562 \times 10^{6} \quad \textbf{(7)}$$

$$Izy := -527698338.200 \cdot 10^{-6}$$

$$Izy := -527.6983382 \quad \textbf{(8)}$$

$$Izz := 5.98623282290516000E14 \cdot 10^{-6}$$

$$Izz := 5.986232823 \times 10^{8} \quad \textbf{(9)}$$

C Delftship hydrostatics report

Design hydrostatics report

Designer	
Created by	
Comment	
Filename	Revolt (Final part ASCI (NO LEAKS)).fbm

Design length	60,000 (m)	Midship location	30,000 (m)
Length over all	60,231 (m)	Relative water density	1,0250
Design beam	14,500 (m)	Mean shell thickness	0,0000 (m)
Maximum beam	14,575 (m)	Appendage coefficient	1,0000
Design draft	5,045 (m)		

Volume properties		Waterplane properties	
Moulded volume	2341,67 (m^3)	Length on waterline	60,185 (m)
Total displaced volume	2341,67 (m^3)	Beam on waterline	14,563 (m)
Displacement	2400,21 (tonnes)	Entrance angle	65,757 (Degr.)
Block coefficient	0,5335	Waterplane area	738,46 (m^2)
Prismatic coefficient	0,6968	Waterplane coefficient	0,8488
Vert. prismatic coefficient	0,6285	Waterplane center of floatation	28,474 (m)
Wetted surface area	977,22 (m^2)	Transverse moment of inertia	11072 (m^4)
Longitudinal center of buoyancy	31,038 (m)	Longitudinal moment of inertia	171971 (m^4)
Longitudinal center of buoyancy	1,724 %		
Vertical center of buoyancy	3,152 (m)		

Midship properties		Initial stability	
Midship section area	56,01 (m^2)	Transverse metacentric height	7,880 (m)
Midship coefficient	0,7657	Longitudinal metacentric height	76,591 (m)

Lateral plane	
Lateral area	230,40 (m^2)
Longitudinal center of effort	33,164 (m)
Vertical center of effort	2,918 (m)

The following layer properties are calculated for both sides of the ship

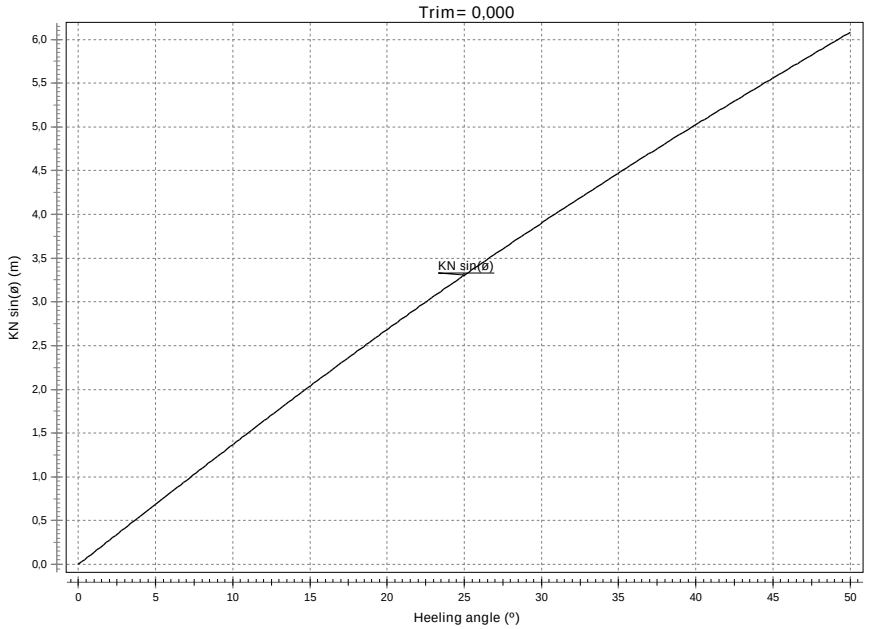
Location	Area (m^2)	Thickness (m)	Weight (tonnes)	LCG (m)	TCG (m)	VCG (m)
Layer 0	2843,52	0,000	0,00	28,314	0,000 (CL)	7,493

NOTE 1: Draft (and all other vertical heights) is measured from base Z=0,000
NOTE 2: All calculated coefficients based on project length, draft and beam.

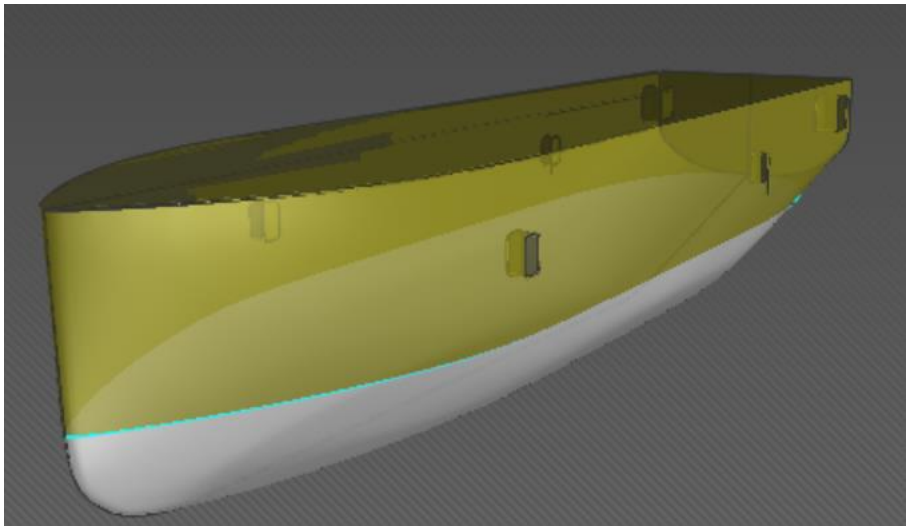
Cross curves

Trim= 0,000

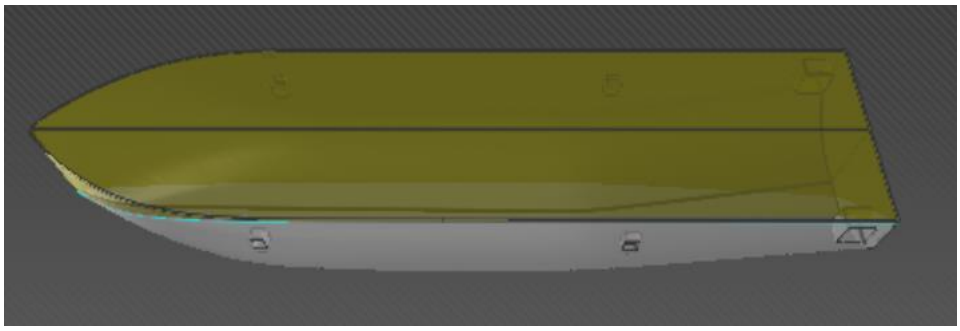
	0,0º	10,0º	20,0º	30,0º	40,0º	50,0º
KN sin(ø)	0,000	1,369	2,684	3,901	5,025	6,080



Heel angle 0 deg



Heel angle 50 deg



D Stability with and without crane

Hydrostatic calculations ReVolt

The ReVolt vessel coeficents from Delfstship

Params:

$p\_sea := 1025$

$$p\_sea := 1025 \quad (1)$$

$g := 9.81$

$$g := 9.81 \quad (2)$$

Initial params from Delftship:

$V := 2341.67$ ## *Displaced water volume*

$$V := 2341.67 \quad (3)$$

$KM\_T := 7.88$

$$KM\_T := 7.88 \quad (4)$$

$KM\_L := 76.591$

$$KM\_L := 76.591 \quad (5)$$

$A\_wl := 738.46$

$$A\_wl := 738.46 \quad (6)$$

Verification that the vessel does not capsize:

$m\_base := 27515.861$

$$m\_base := 27515.861 \quad (7)$$

$m\_link1 := 13648$

$$m\_link1 := 13648 \quad (8)$$

$m\_link2 := 1416$

$$m\_link2 := 1416 \quad (9)$$

$m\_ship := 2400 \cdot 10^3$

$$m\_ship := 2400000 \quad (10)$$

$m\_spreader := 1.5 \cdot 10^3$

$$m\_spreader := 1500.0 \quad (11)$$

$m\_container := 5 \cdot 10^3$

$$m\_container := 5000 \quad (12)$$

$L1 := 19.201$

$$L1 := 19.201 \quad (13)$$

$L2 := 6.24$

$$L2 := 6.24 \quad (14)$$

$Lcg\_link1 := 8.31$

$$Lcg\_link1 := 8.31 \quad (15)$$

$Lcg\_link2 := 2.727$

$$Lcg\_link2 := 2.727 \quad (16)$$

$H\_ship := 12.5$

$$H\_ship := 12.5 \quad (17)$$

$vcg\_ship\_delftship := 7.493$

$$vcg\_ship\_delftship := 7.493 \quad (18)$$

Checking loaded crane in worst configuration while loaded with 5 tonn container

$numenator := (m\_base \cdot H\_ship + m\_link1 \cdot (H\_ship + Lcg\_link1) + m\_link2 \cdot (H\_ship + L1 + Lcg\_link2) + (m\_spreader + m\_container) \cdot (H\_ship + L1 + L2) + m\_ship \cdot vcg\_ship\_delftship) :$

$denominator := m\_base + m\_spreader + m\_container + m\_link1 + m\_link2 + m\_ship :$

$cog\_ship\_crane := \frac{numenator}{denominator}$

$$cog\_ship\_crane := 7.719850214 \quad (19)$$

$GM\_T\_worstCase := KM\_T - cog\_ship\_crane$

$$GM\_T\_worstCase := 0.160149786 \quad (20)$$

Finding restoring coefficient for roll:

$GM\_T\_noCrane := KM\_T - vcg\_ship\_delftship$

$$GM\_T\_noCrane := 0.387 \quad (21)$$

$C\_44\_worst := p\_sea \cdot g \cdot V \cdot GM\_T\_worstCase$

$$C\_44\_worst := 3.770899236 \times 10^6 \quad (22)$$

$C\_44\_nocrane := p\_sea \cdot g \cdot V \cdot GM\_T\_noCrane$

$$C\_44\_nocrane := 9.112331903 \times 10^6 \quad (23)$$

$$GM\_mean := \frac{(GM\_T\_worstCase + GM\_T\_noCrane)}{2}$$

$$GM\_mean := 0.2735748930 \quad \textbf{(24)}$$

$$C\_44\_mean := p\_sea \cdot g \cdot V \cdot GM\_mean$$

$$C\_44\_mean := 6.441615570 \times 10^{6} \quad \textbf{(25)}$$

Finding restoring coefficent for pitch:

$$GM\_L := KM\_L - vcg\_ship\_delftship$$

$$GM\_L := 69.098 \quad \textbf{(26)}$$

$$GM\_L\_worst := KM\_L - cog\_ship\_crane$$

$$GM\_L\_worst := 68.87114979 \quad \textbf{(27)}$$

$$C\_55 = p\_sea \cdot g \cdot V \cdot GM\_L$$

$$C\_55 = 1.626986847 \times 10^{9} \quad \textbf{(28)}$$

$$GM\_L\_mean := \frac{(GM\_L + GM\_L\_worst)}{2}$$

$$GM\_L\_mean := 68.98457490 \quad \textbf{(29)}$$

$$C\_55\_mean = p\_sea \cdot g \cdot V \cdot GM\_L\_mean$$

$$C\_55\_mean = 1.624316131 \times 10^{9} \quad \textbf{(30)}$$

Finding restoring force in heave:

#Faltinsen

$$C\_33\_F = A\_wl \cdot p\_sea \cdot g$$

$$C\_33\_F = 7.425399915 \times 10^{6} \quad \textbf{(31)}$$

 # Finding restoring coefficent given draft of 5.045 from delftship

$$C\_33 = \frac{m\_ship \cdot g}{5.045}$$

$$C\_33 = 4.666798811 \times 10^{6} \quad \textbf{(32)}$$

E Point-to-point coefficients

Point-to-point motion finding a0, a1, a2, a3, a4, a5

restart :

$q := a5 \cdot t^5 + a4 \cdot t^4 + a3 \cdot t^3 + a2 \cdot t^2 + a1 \cdot t + a0$

$$q := a5\, t^5 + a4\, t^4 + a3\, t^3 + a2\, t^2 + a1\, t + a0 \quad (1)$$

$q\_dot := diff(q, t)$

$$q\_dot := 5\, a5\, t^4 + 4\, a4\, t^3 + 3\, a3\, t^2 + 2\, a2\, t + a1 \quad (2)$$

$q\_ddot := diff(q\_dot, t)$

$$q\_ddot := 20\, a5\, t^3 + 12\, a4\, t^2 + 6\, a3\, t + 2\, a2 \quad (3)$$

#Finding initial position, valocity and accel terms

$eq1 := qi = eval(q, t = 0)$

$$eq1 := qi = a0 \quad (4)$$

$eq2 := qi\_dot = eval(q\_dot, t = 0)$

$$eq2 := qi\_dot = a1 \quad (5)$$

$eq3 := qi\_ddot = eval(q\_ddot, t = 0)$

$$eq3 := qi\_ddot = 2\, a2 \quad (6)$$

#Finding end position, valocity and accel terms

$eq4 := qf = eval(q, t = tf)$

$$eq4 := qf = a5\, tf^5 + a4\, tf^4 + a3\, tf^3 + a2\, tf^2 + a1\, tf + a0 \quad (7)$$

$eq5 := qf\_dot = eval(q\_dot, t = tf)$

$$eq5 := qf\_dot = 5\, a5\, tf^4 + 4\, a4\, tf^3 + 3\, a3\, tf^2 + 2\, a2\, tf + a1 \quad (8)$$

$eq6 := qf\_ddot = eval(q\_ddot, t = tf)$

$$eq6 := qf\_ddot = 20\, a5\, tf^3 + 12\, a4\, tf^2 + 6\, a3\, tf + 2\, a2 \quad (9)$$

Solving for a1,a2,a3,a4,a5

$solve(\{eq1, eq2, eq3, eq4, eq5, eq6\}, \{a0, a1, a2, a3, a4, a5\})$

$$\left\{ a0 = qi,\ a1 = qi\_dot,\ a2 = \frac{qi\_ddot}{2},\ a3 \right. \quad (10)$$

$$= \frac{qf\_ddot\, tf^2 - 3\, tf^2\, qi\_ddot - 8\, qf\_dot\, tf - 12\, qi\_dot\, tf + 20\, qf - 20\, qi}{2\, tf^3},\ a4 =$$

$$\left\{ -\frac{2\,qf\_ddot\,tf^2 - 3\,tf^2\,qi\_ddot - 14\,qf\_dot\,tf - 16\,qi\_dot\,tf + 30\,qf - 30\,qi}{2\,tf^4},\ a5 \right.$$
$$\left. = \frac{qf\_ddot\,tf^2 - tf^2\,qi\_ddot - 6\,qf\_dot\,tf - 6\,qi\_dot\,tf + 12\,qf - 12\,qi}{2\,tf^5} \right\}$$

F Code from all physical elements in bond graph

F.1 Ship and crane parameters

```
parameters

    real global g= 9.81; //Gravitational Constant
    real scalingFactor = 6.5;        //Scaling factor
    real global r_tip_container = 15; // desired distance from crane-tip and top of contanienr
    real global P_desired_container[3] = [0;20;7.69]; // Desired container position


    // Ship parameters
    real global H = 12.5;                                   // [m] Height of vesssel
    real global B = 14.4;                                   // [m] (Width)
    real global Draught = 5.045;                // [m] Draught of vessel
    real global Lpp = 56.82;                         // [m]                        (Length between perpendiculars)
    real global Mass = 2400000;                 // [kg]                 (Ship weight)
    real global Rho_w = 1025;        // [kg/m^3] (Sea water density)
    real global vcg_ship = 7.493; // [m] Vertical centre of gravity of ship without crane from delftship


    // Hydrostatic restoring matrix calculated in Maple
    real global C[6,6] =         [       0.0,                        0.00,
                                0.00,                          0.00,
                                0.00,                          0.00;
                                                              0.00,
                                                                                    0.0,
                                                                                    0.00,
                                                                                    0.00,
                                                                                    0.00,
                                                                                    0.00;
                                                              0.00,
                                                                                    0.00,
                                                                                    46667
                                                                    98.811,         0.00,
                                                                                    0.0,
                                                                                0.00;
                                                              0.00,
                                                                                    0.00,
                                                                                    0.00,
                                                                                     6.44
                                                                    1615570*10^6,
                                                                                0.00,
                                                                                    0.0;
```

```
28                                                  0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                       1.624
                                                    ↪  540534*10^9,
                                                    ↪                  0.00;
29                                                  0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.0];
30
31          // Tuned Damping matrix
32          real global D[6,6] =         10^2*[        150160.00*10,         0.00,
            ↪                          0.00,                 0.00,                       0.00,
            ↪                  0.00;
33                                                  0.00,                        286
                                                    ↪  000.0,         0.00,
                                                    ↪                  0.00,
                                                    ↪                       0.00,
                                                    ↪                  0.00;
34                                                  0.00,                        0.00
                                                    ↪  ,
                                                    ↪                       1e7,
                                                    ↪                  0.00,
                                                    ↪                       0.00,
                                                    ↪                  0.00;
35                                                  0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                  3*128000.0,
                                                    ↪                  0.00,
                                                    ↪                  0.00;
36                                                  0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                  0.00,
                                                    ↪                       1e7*1
                                                    ↪  0,
                                                    ↪                  0.00;
37                                                  0.00,
                                                    ↪                       0.00,
                                                    ↪                       0.00,
                                                    ↪                  0.00,
                                                    ↪                       0.00,
                                                    ↪                  161440.0];
38
39
40
41
42
43  variables
44
45
46  // Vessel initial angels
47          real global phi_rad_init,theta_rad_init,psi_rad_init;
```

```
48
49  //Crane parameters //
50          real global L1,L2,b,a,h,w,c,u,r,s,k,n,o,p,Xcm1,Ycm1;
51          real global m1,m2,m3;
52          real global theta1_init_deg,theta2_init_deg,theta3_init_deg;
53          real global theta1_rad_init,theta2_rad_init,theta3_rad_init;
54
55  // Crane link vectors
56          real global rb_1_b[3];
57          real global r1_2_1[3];
58          real global r2_3_2[3];
59          real global r3_8_3[3];
60
61  initialequations
62          //Parameters for crane arm
63          L1= 2.954*scalingFactor;                L2= 0.96*scalingFactor;                b=
            ↪  0.445*scalingFactor;                         a= 0.1*scalingFactor;
64          h= 0.495*scalingFactor;                w= 2.150*scalingFactor;                c=
            ↪  0.19*scalingFactor;                          u=
            ↪  0.280*scalingFactor;
65          r= 0.100*scalingFactor;                s= 0.118*scalingFactor;                k=
            ↪  1.286*scalingFactor;                         n=
            ↪  0.420*scalingFactor;
66          o= 0.324*scalingFactor;                p= 0.429*scalingFactor;                Xcm1 =
            ↪  0*scalingFactor;                          Ycm1 = -0.023*scalingFactor;
67
68          m1 = 27515.861 ;          // [kg] mass of base
69          m2 = 13648.269 ;          // [kg] mass of lower arm (link1)
70          m3 = 1416.126 ;  // [kg] mass of upper arm (link2)
71
72          theta1_init_deg = 0;
73          theta2_init_deg = 50;
74          theta3_init_deg = -57;
75
76          // Crane vectors to find crane tip
77          rb_1_b = [0; 0; 0]; r1_2_1 = [0; 0; h]; r2_3_2 = [0; L1; 0]; r3_8_3 = [0; L2; 0];
78
79
80          theta1_rad_init = theta1_init_deg*pi/180;
81          theta2_rad_init = theta2_init_deg*pi/180;
82          theta3_rad_init = theta3_init_deg*pi/180;
83
84
85
```

F.2 Spreader and wire parameters

```
1       parameters
2           // Spreader params
3
4           // Dimensions
5           real global H_spreader = 0.3048;                        // [m] Height
6           real global B_spreader = 2.4384;                        // [m]
            ↪  Width
7           real global L_spreader = 6.096;                                 // [m] Length
8           real global MassSpreader = 1500;                // [kg] Weight
```

```
        real global initalWireLength = 5; // Initial length of spreader line [m]
        real global E_wire= 200000000000; //Wire E-module [Pa]
        real global D_wire= 0.002;//Diameter of wire [m]
        real global r_CG_Spreader [3,1] = [0;0;0]; // centre of gravity relative to geometric
        ↪  centre

// Inertia calculations from Maple
        real I_44 = 1.3272e8;
        real I_55 = 7.3782e8;
        real I_66 = 6.8804e8;
        real I_46 = 7.2000e6;

variables
        real hidden zc;
        real global M[6,6];
        real global Minv[6,6];

initialequations

        zc = r_CG_Spreader[3];

         M =                          [        MassSpreader,                                  0.00,
         ↪                                                          0.00,
         ↪                                  0.00,
         ↪                                  MassSpreader*zc,                0.00;
                                                                    0.00,                          ⌋
                                                                    ↪          MassSpreader,
                                                                    ↪                          ⌋
                                                                    ↪                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                          ⌋
                                                                    ↪          -MassSpreader*zc,
                                                                    ↪                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                  0.00;
                                                                    0.00,                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                          ⌋
                                                                    ↪                          ⌋
                                                                    ↪          MassSpreader,
                                                                    ↪                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                          ⌋
                                                                    ↪          0.0,
                                                                    ↪                                0.00;
                                                                    0.00,                          ⌋
                                                                    ↪          -MassSpreader*zc,
                                                                    ↪                          ⌋
                                                                    ↪                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                          ⌋
                                                                    ↪          I_44,
                                                                    ↪                          ⌋
                                                                    ↪          0.00,
                                                                    ↪                  -I_46;
```

```
                                                    MassSpreader*zc,                    ]
                                                    ↪                   0.00,
                                                    ↪                                   ]
                                                    ↪                                   ]
                                                    ↪           0.00,
                                                    ↪                                   ]
                                                    ↪           0.00,
                                                    ↪                                   ]
                                                    ↪           I_55,                   ]
                                                    ↪           0.00;
                                                    0.00,                               ]
                                                    ↪           0.00,
                                                    ↪                                   ]
                                                    ↪                                   ]
                                                    ↪           0.00,
                                                    ↪                                   ]
                                                    ↪           -I_46,
                                                    ↪                                   ]
                                                    ↪           0.00,
                                                    ↪                   I_66];


        Minv = inverse(M);


```

F.3 IC ship-crane

```
parameters

 string dll_name = 'Ship-crane-integration.dll';
 real initialTransformAngles[3] = [-pi;0;0];
 real global H;
 real global Draught;
 real global vcg_ship;

variables


  real initialWaterlinePosition;
 // quasi coordinate vector
         real omega[9];
        real vx, vy, vz;      // translational velocity of vessel
        real phi, theta, psi; // Rotations of vessel
        real theta1, theta2, theta3; // Crane joint rotations
         real phi_dot, theta_dot, psi_dot, theta1_dot, theta2_dot, theta3_dot;
   real EulerAngles[3];
        real EulerAngles_dot[3];


 // State space terms

        real fp[9];
        real omega_T_dBdq[9,9];
```

```
28
29
30          real e_in[6];
31          real e_out[9];
32          real f_out[6];
33
34    // trasformation matrixes
35          real craneTip[3];
36          real craneBase[3];
37          real craneLink1[3];
38          real craneLink2[3];
39          real global r0_b_cog[3];
40          real global r0_cog_0[3];
41          real r0_cog_0_viz[3];
42          real r0_b_cog_viz[3];
43          real cog_body[3];
44          real global Rb_0[3,3], Rb_1[3,3], R1_2[3,3], R2_3[3,3];
45          // Frame  vizualization
46          real revoltBottomPosition[3];
47          real BodyFrame[3,3];
48          real BaseFrame[3,3];
49          real Link1Frame[3,3];
50          real Link2Frame[3,3];
51          real CraneTipFrame[3,3];
52
53          real global rb_1_b[3];
54          real global r1_2_1[3];
55          real global r2_3_2[3];
56          real global r3_8_3[3];
57          real global theta1_rad_init,theta2_rad_init,theta3_rad_init;
58           real global beta[9,9];
59           real global beta_T[9,9];
60           real T_mat[3,3];
61
62          real global initialVesselAngles[3];
63    //Mass matrix B
64    real B_tot[9,9];
65    real B_in[6],B_out[45];
66
67    // Added mass matrix coriolies matrix from added mass
68    real A[9,9];
69    real B_A[9,9];
70    real C_A[9,9];
71    real Xu;
72    real Yv;
73    real Zw;
74    real Kp;
75    real Mq;
76    real Nr;
77    //Partial derivatives of B
78
79    real dBdq1[9,9];
80    real dBdq2[9,9];
81    real dBdq3[9,9];
82    real dBdq4[9,9];
83    real dBdq5[9,9];
84    real dBdq6[9,9];
85
```

```
 real dBdq7[9,9];
 real dBdq8[9,9];
 real dBdq9[9,9];

 real dBdq7_out[45];
 real dBdq8_out[45];
 real dBdq9_out[45];

// alpha matrix parial derivatives
 real alpha[9,9];
 real dalpha_dtheta_mat[9,9];
 real dalpha_dphi_mat[9,9];
 real dalpha_dpsi_mat[9,9];

 real dalpha_dq1[9,9];
 real dalpha_dq2[9,9];
 real dalpha_dq3[9,9];
 real dalpha_dq7[9,9];
 real dalpha_dq8[9,9];
 real dalpha_dq9[9,9];

 real alpha_in[3];

 real dalpha_dtheta_out[81];
 real dalpha_dphi_out[81];
 real dalpha_dpsi_out[81];

// gamma
 real Xi[9,9];
 real daplha_ij_dq[9];
 real gamma[9,9];
 real gamma_right[9,9];

 // Iteration variables
 real i,j;
 real counter, turn_counter;

 // Intial angles
 real init_phi_rad;
 real init_theta_rad;
 real init_psi_rad;

initialequations



        initialVesselAngles = [0;0;0] + initialTransformAngles;

        init_phi_rad = initialVesselAngles[1];
        init_theta_rad = initialVesselAngles[2];
        init_psi_rad = initialVesselAngles[3];
// System mass matrix of marine vessel is not a
// function of generalized coord.
        dBdq1[1:9,1:9] = 0;
         dBdq2[1:9,1:9] = 0;
   dBdq3[1:9,1:9] = 0;
   dBdq4[1:9,1:9] = 0;
   dBdq5[1:9,1:9] = 0;
```

```
144     dBdq6[1:9,1:9] = 0;
145
146 // alpha matrix is only dependent on vessel rotations: phi, theta and psi
147         dalpha_dq1[1:9,1:9] = 0;
148          dalpha_dq2[1:9,1:9] = 0;
149          dalpha_dq3[1:9,1:9] = 0;
150          dalpha_dq7[1:9,1:9] = 0;
151          dalpha_dq8[1:9,1:9] = 0;
152          dalpha_dq9[1:9,1:9] = 0;
153
154
155         counter = 0;
156         turn_counter = 0;
157
158
159         // From {b} to {0}
160         Rb_0 = [cos(init_psi_rad)*cos(init_theta_rad), sin(init_psi_rad)*cos(init_theta_rad),
        ↪  -sin(init_theta_rad);
161                                 -sin(init_psi_rad)*cos(init_phi_rad) +
                                    ↪  cos(init_psi_rad)*sin(init_theta_rad)*sin(init_phi_rad),
                                    ↪  cos(init_psi_rad)*cos(init_phi_rad) +
                                    ↪  sin(init_psi_rad)*sin(init_theta_rad)*sin(init_phi_rad),
                                    ↪  cos(init_theta_rad)*sin(init_phi_rad);
162                                 sin(init_psi_rad)*sin(init_phi_rad) +
                                    ↪  cos(init_psi_rad)*sin(init_theta_rad)*cos(init_phi_rad),
                                    ↪  -cos(init_psi_rad)*sin(init_phi_rad) +
                                    ↪  sin(init_psi_rad)*sin(init_theta_rad)*cos(init_phi_rad),
                                    ↪  cos(init_theta_rad)*cos(init_phi_rad)];
163
164         // From {1} to {b}
165         Rb_1 = [sin(theta1_rad_init ), cos(theta1_rad_init ), 0;
166                         cos(theta1_rad_init ), -sin(theta1_rad_init ), 0;
167                         0, 0, -1];
168
169         // From {2} to {1}
170         R1_2 = [1, 0, 0;
171                          0, cos(theta2_rad_init), -sin(theta2_rad_init);
172                           0, sin(theta2_rad_init), cos(theta2_rad_init)];
173         // From {3} to {2}
174         R2_3 = [1, 0, 0;
175                0, cos(theta3_rad_init), -sin(theta3_rad_init);
176                0, sin(theta3_rad_init), cos(theta3_rad_init)];
177
178         craneTip = transpose(Rb_0)*rb_1_b + transpose(Rb_0)*transpose(Rb_1)*r1_2_1 +
        ↪  transpose(Rb_0)*transpose(Rb_1)*R1_2*r2_3_2 +
        ↪  transpose(Rb_0)*transpose(Rb_1)*R1_2*R2_3*r3_8_3;
179
180         //Constructing added mass matrix from conevntional ship params
181         B_A = 0;
182         B_A[1,1] = 1.4e6; B_A[2,2] = 7.2e6; B_A[3,3] = 11.1e7;
183         B_A[4,4] = 1.75e8; B_A[5,5] = 8e10;  B_A[6,6] = 4.2e9;
184
185 equations
186
187
188
189         // Rotation matrices
190         Rb_0 = [cos(psi)*cos(theta), sin(psi)*cos(theta), -sin(theta);
```

```
191                                 -sin(psi)*cos(phi) + cos(psi)*sin(theta)*sin(phi),
                                    ↪  cos(psi)*cos(phi) + sin(psi)*sin(theta)*sin(phi),
                                    ↪  cos(theta)*sin(phi);
192                                 sin(psi)*sin(phi) + cos(psi)*sin(theta)*cos(phi),
                                    ↪  -cos(psi)*sin(phi) + sin(psi)*sin(theta)*cos(phi),
                                    ↪  cos(theta)*cos(phi)];
193
194         T_mat = [1, sin(phi)*tan(theta), cos(phi)*tan(theta);
195                                 0, cos(phi), -sin(phi);
196                                 0, sin(phi)* 1/cos(theta), cos(phi)* 1/cos(theta)];
197
198
199
200
201         // Model only recives input forces on the vessel, as the crane joint velocities are
            ↪  governed by actuators
202         e_in[1] = vI.e[1]; e_in[2]=vI.e[2]; e_in[3]=vI.e[3];
203         e_in[4] = wI.e[1]; e_in[5] = wI.e[2]; e_in[6] = wI.e[3];
204
205         // Setting input velocities
206         EulerAngles_dot=T_mat*wC.f;
207         EulerAngles = int(T_mat*wC.f,initialVesselAngles);
208
209     vx = vC.f[1]; vy=vC.f[2]; vz=vC.f[3];
210         phi = EulerAngles[1]; theta = EulerAngles[2]; psi = EulerAngles[3];
211         phi_dot = EulerAngles_dot[1]; theta_dot = EulerAngles_dot[2]; psi_dot =
            ↪  EulerAngles_dot[3];
212         theta1 = jointAngles[1]; theta2 = jointAngles[2]; theta3 = jointAngles[3];
213         theta1_dot = qc_C.f[1]; theta2_dot = qc_C.f[2]; theta3_dot = qc_C.f[3];
214
215         // Creating quasi-coordinate vector
216         omega= [vx;vy;vz ;wC.f[1];wC.f[2];wC.f[3]; theta1_dot; theta2_dot; theta3_dot];
217
218     // Setting input arguments to .dll functions
219         r0_cog_0 = transpose(Rb_0) * int(vC.f) + [0;0;vcg_ship-Draught];
220         cog_body = Rb_0 * r0_cog_0;
221         B_in = [theta1; theta2; theta3; cog_body[1];cog_body[2];cog_body[3]];
222         alpha_in = [phi;theta;psi];
223
224         // Exporting outputs from .dll
225         B_out = dll(dll_name, 'updatedB', B_in);
226         dBdq7_out = dll(dll_name, 'update_dBdq7', B_in);
227         dBdq8_out = dll(dll_name, 'update_dBdq8', B_in);
228         dBdq9_out = dll(dll_name, 'update_dBdq9', B_in);
229
230         dalpha_dtheta_out = dll(dll_name, 'dalpha_dtheta', alpha_in);
231          dalpha_dphi_out = dll(dll_name, 'dalpha_dphi', alpha_in);
232          dalpha_dpsi_out = dll(dll_name, 'dalpha_dpsi', alpha_in);
233
234         // Creating the matrixes B_tot, dBdq7, dBdq8, dBdq9 from .dll outputs
235         if counter < 45 then
236                 for i = 1 to 9 do
237                         for j = i to 9 do
238                                 counter =  counter + 1;
239                                 B_tot[i,j] = B_out[counter];
240                                 B_tot[j,i] = B_tot[i,j];
241                         end;
242                 end;
```

```
243         counter = 0;
244         end;
245 
246         if counter < 45 then
247                  for i = 1 to 9 do
248                         for j = i to 9 do
249                                 counter =  counter + 1;
250                                 dBdq7[i,j] = dBdq7_out[counter];
251                                 dBdq7[j,i] = dBdq7[i,j];
252                         end;
253                  end;
254         counter = 0;
255         end;
256 
257         if counter < 45 then
258                  for i = 1 to 9 do
259                         for j = i to 9 do
260                                 counter =  counter + 1;
261                                 dBdq8[i,j] = dBdq8_out[counter];
262                                 dBdq8[j,i] = dBdq8[i,j];
263                         end;
264                  end;
265                 counter = 0;
266         end;
267 
268         if counter < 45 then
269                  for i = 1 to 9 do
270                         for j = i to 9 do
271                                 counter =  counter + 1;
272                                 dBdq9[i,j] = dBdq9_out[counter];
273                                 dBdq9[j,i] = dBdq9[i,j];
274                         end;
275                  end;
276                 counter = 0;
277         end;
278 
279         // Exporting dalpha_dphi, dalpha_dtheta, dalpha_dpsi from .dll output
280 
281         if counter < 81 then
282                  for i = 1 to 9 do
283                          for j = 1 to 9 do
284                                  counter =  counter + 1;
285                                  dalpha_dphi_mat[i,j] = dalpha_dphi_out[counter];
286                         end;
287                  end;
288                 counter = 0;
289         end;
290 
291         if counter < 81 then
292                  for i = 1 to 9 do
293                          for j = 1 to 9 do
294                                  counter =  counter + 1;
295                                  dalpha_dtheta_mat[i,j] = dalpha_dtheta_out[counter];
296                         end;
297                  end;
298                 counter = 0;
299         end;
300 
```

```
301
302         if counter < 81 then
303                 for i = 1 to 9 do
304                         for j = 1 to 9 do
305                                 counter =  counter + 1;
306                                 dalpha_dpsi_mat[i,j] = dalpha_dpsi_out[counter];
307                         end;
308                 end;
309                counter = 0;
310         end;
311
312         // Creating the beta trasnformation matix
313         beta[1:3,1:3] = Rb_0;
314         beta[4:6,4:6] = T_mat;
315         beta[7:9,7:9] = eye(3);
316         beta[1:3,4:9] = 0;
317         beta[4:9,1:3] = 0;
318         beta[7:9,1:6] = 0;
319         beta[3:6,7:9] = 0;
320         beta_T = transpose(beta);
321
322         //Creating gamma
323         if counter < 81 then
324                 for i = 1 to 9 do
325                         for j = 1 to 9 do
326                                 counter =  counter + 1;
327                                daplha_ij_dq = [0;0;0;dalpha_dphi_mat[i,j];dalpha_dtheta_mat[i,j]
                                ↪  ;dalpha_dpsi_mat[i,j];0;0;0];
328                                 Xi[i,j] = transpose(omega)*transpose(beta)*daplha_ij_dq;
329                         end;
330                 end;
331                counter = 0;
332         end;
333
334     gamma = Xi;
335
336
337         gamma_right[1,1:9] = transpose(omega)*beta_T* dalpha_dq1;
338         gamma_right[2,1:9] = transpose(omega)*beta_T*dalpha_dq2 ;
339         gamma_right[3,1:9] = transpose(omega)*beta_T*dalpha_dq3;
340         gamma_right[4,1:9] = transpose(omega)*beta_T*dalpha_dphi_mat;
341         gamma_right[5,1:9] = transpose(omega)*beta_T*dalpha_dtheta_mat;
342         gamma_right[6,1:9] = transpose(omega)*beta_T*dalpha_dpsi_mat;
343         gamma_right[7,1:9] = transpose(omega)*beta_T*dalpha_dq7;
344         gamma_right[8,1:9] = transpose(omega)*beta_T*dalpha_dq8;
345         gamma_right[9,1:9] = transpose(omega)*beta_T*dalpha_dq9;
346
347     gamma = Xi - gamma_right;
348
349
350         //Finding fp(q, omega)
351
352         omega_T_dBdq[1,1:9] = transpose(omega)*dBdq1;
353          omega_T_dBdq[2,1:9] = transpose(omega)*dBdq2;
354          omega_T_dBdq[3,1:9] = transpose(omega)*dBdq3;
355          omega_T_dBdq[4,1:9] = transpose(omega)*dBdq4;
356          omega_T_dBdq[5,1:9] = transpose(omega)*dBdq5;
357          omega_T_dBdq[6,1:9] = transpose(omega)*dBdq6;
```

```
 omega_T_dBdq[7,1:9] = transpose(omega)*dBdq7;
 omega_T_dBdq[8,1:9] = transpose(omega)*dBdq8;
 omega_T_dBdq[9,1:9] = transpose(omega)*dBdq9;

// Coriolies matrix from added mass

C_A[1:9,1:9] = 0;
C_A[1:3,4:6] = -skew(B_A[1:3,1:3]*vC.f+B_A[1:3,4:6]*wC.f);
C_A[4:6,1:3] = -skew(B_A[1:3,1:3]*vC.f+B_A[1:3,4:6]*wC.f);
C_A[4:6,4:6] = -skew(B_A[4:6,1:3]*vC.f+B_A[4:6,4:6]*wC.f);

fp = -beta_T*gamma*B_tot*omega + 0.5*beta_T*omega_T_dBdq*omega + C_A*omega;
e_out = fp;
f_out = inverse(B_tot[1:6,1:6] + B_A[1:6,1:6])*int(e_in);

// Setting output velocities as a result of input forces
vI.f[1] = f_out[1]; vI.f[2] = f_out[2]; vI.f[3] = f_out[3];
wI.f[1] = f_out[4]; wI.f[2] = f_out[5]; wI.f[3] = f_out[6];

// Setting output inertia force as a result of the input velocities
vC.e[1] = e_out[1]; vC.e[2] = e_out[2]; vC.e[3] = e_out[3];
wC.e[1] = e_out[4]; wC.e[2] = e_out[5]; wC.e[3] = e_out[6];
qc_C.e[1] = e_out[7]; qc_C.e[2] = e_out[8]; qc_C.e[3] = e_out[9];



//Finding transformation matrices to find crane tip

Rb_1 = [sin(theta1), cos(theta1), 0;
                cos(theta1), -sin(theta1), 0;
                0, 0, -1];

R1_2 = [1, 0, 0;
                 0, cos(theta2), -sin(theta2);
                  0, sin(theta2), cos(theta2)];
R2_3 = [1, 0, 0;
       0, cos(theta3), -sin(theta3);
       0, sin(theta3), cos(theta3)];


//Frame locations and crane tip

r0_b_cog = [sin(theta)*(H - vcg_ship); sin(phi)*(H - vcg_ship);(H - vcg_ship)] + r0_cog_0;
craneBase = r0_b_cog + transpose(Rb_0)*rb_1_b;
craneLink1 = r0_b_cog + transpose(Rb_0)*rb_1_b + transpose(Rb_0)*transpose(Rb_1)*r1_2_1;
craneLink2 = r0_b_cog + transpose(Rb_0)*rb_1_b + transpose(Rb_0)*transpose(Rb_1)*r1_2_1 +
↪  transpose(Rb_0)*transpose(Rb_1)*R1_2*r2_3_2;
craneTip = r0_b_cog + transpose(Rb_0)*rb_1_b + transpose(Rb_0)*transpose(Rb_1)*r1_2_1 +
↪  transpose(Rb_0)*transpose(Rb_1)*R1_2*r2_3_2 +
↪  transpose(Rb_0)*transpose(Rb_1)*R1_2*R2_3*r3_8_3;

//Visualization
initialWaterlinePosition = 0;
revoltBottomPosition = r0_cog_0 - [sin(theta)*(vcg_ship); sin(phi)*(vcg_ship);(vcg_ship)];
r0_cog_0_viz = r0_cog_0;
r0_b_cog_viz = r0_b_cog;


```

```
        // Frame  vizualization
        BodyFrame = Rb_0; // From {b} to {0}
        BaseFrame = Rb_0*Rb_1; // From {1} to {0}
        Link1Frame = Rb_0*Rb_1*R1_2; // From {2} to {0}
        Link2Frame = Rb_0*Rb_1*R1_2*R2_3; // From {3} to {0}
        CraneTipFrame =  Rb_0*Rb_1*R1_2*R2_3;

        craneTip_signal = craneTip;
```

F.4 MTF - world-to-body-transform

```
variables
 real global beta[9,9];
 real global beta_T[9,9];


equations
        p1.e = beta[4:6,4:6] * p2.e;
        p2.f = beta_T[4:6,4:6] * p1.f;
```

F.5 R-translational

```
parameters
        real global D[6,6];
equations
        p.e = D[1:3,1:3] * p.f;
```

F.6 R-rotational

```
parameters
        real global D[6,6];
equations
        p.e = D[4:6,4:6] * p.f;
```

F.7 C-translational

```
parameters
        real global C[6,6];
        real init [3,1] = [0;0;-5.045];

equations

    state = int(p.f,init);
```

```
    p.e = C[1:3,1:3] *state;


```

F.8 C-rotational

```
parameters
        real global C[6,6];
equations
    state = int(p.f);
    p.e = C[4:6,4:6] *state;

```

F.9 Se - ship gravity

```
parameters
        real global Mass;
        real global g;
        real effort[3,1] = [0; 0;-Mass*g];
variables
        real flow [3];
equations
        p.e = effort;
        flow = p.f;

```

G Gravity from crane links and spreader

G.1 Se-base

```
parameters
        real global g;
variables
        real flow[6];
        real global m1,m2,m3;
equations
        p.e = 0;
        p.e[3] = -m1*g;
        flow = p.f;

```

G.2 Se-link1

```
parameters
        real global g;
variables
        real flow[6];
        real global m1,m2,m3;
equations
        p.e = 0;
```

```
        p.e[3] = -m2*g;
        flow = p.f;
```

G.3 Se-link2

```
parameters
        real global g;
variables
        real flow[6];
        real global m1,m2,m3;
equations
        p.e = 0;
        p.e[3] = -m3*g;
        flow = p.f;
```

G.4 Se-spreader

```
parameters
        real global MassSpreader;
        real global g;
variables
        real flow [3];
        real effort[3,1];
equations
        effort = [0; 0; -MassSpreader * g];
        p.e = effort;
        flow = p.f;

```

G.5 MTF Jcm-base

```
variables
        real Jcm_base[6,9], test[9,6], theta1;
        real global Ycm1,Xcm1;
        real global Rb_0[3,3];
        real a[6];
        real b[6];
equations

        theta1 = jointAngles[1];
        Jcm_base = [1, 0, 0, 0, 0, -cos(theta1)*Xcm1 + sin(theta1)*Ycm1, cos(theta1)*Xcm1 -
        ↪  sin(theta1)*Ycm1, 0, 0;
                                    0, 1, 0, 0, 0, sin(theta1)*Xcm1 + cos(theta1)*Ycm1,
                                    ↪  -sin(theta1)*Xcm1 - cos(theta1)*Ycm1, 0, 0;
                                        0, 0, 1, cos(theta1)*Xcm1 - sin(theta1)*Ycm1,
                                        ↪  -sin(theta1)*Xcm1 - cos(theta1)*Ycm1, 0, 0, 0, 0;
                                        0, 0, 0, 1, 0, 0, 0, 0, 0;
                                        0, 0, 0, 0, 1, 0, 0, 0, 0;
                                        0, 0, 0, 0, 0, 1, -1, 0, 0];


        a[1:3] = Rb_0 * p2.e[1:3];
        a[4:6] = Rb_0 * p2.e[4:6];
```

```
        p1.e = transpose(Jcm_base) *p2.e;


        b = Jcm_base * p1.f;

        p2.f[1:3] = transpose(Rb_0)* b[1:3];
        p2.f[4:6] = transpose(Rb_0)* b[4:6];

```

G.6 MTF Jcm-link1

```
variables
        real Jcm_link1[6,9], theta1, theta2;
        real global Ycm1,Xcm1,k, h;
        real a[6],b[6];
        real global Rb_0[3,3];
equations

        theta1 = jointAngles[1];
        theta2 = jointAngles[2];


        a[1:3] = Rb_0 * p2.e[1:3];
        a[4:6] = Rb_0 * p2.e[4:6];

        Jcm_link1 = [1,0,0,0,-h-sin(theta2)* k,sin(theta1)* cos(theta2)* k,-sin(theta1)*
        ↪  cos(theta2)* k,-cos(theta1) * sin(theta2)* k,0;
                                    0,1,0,h+sin(theta2)* k,0,cos(theta1)* cos(theta2)*
                                    ↪  k,-cos(theta1)* cos(theta2) *k,sin(theta1)*
                                    ↪  sin(theta2)* k,0;
                                    0,0,1,-sin(theta1)* cos(theta2) *k,-cos(theta1)*
                                    ↪  cos(theta2)* k,0,0,-(sin(theta1))^2 * cos(theta2)*
                                    ↪  k-(cos(theta1))^2 * cos(theta2)* k,0;
                                    0,0,0,1,0,0,0,sin(theta1),0;
                                    0,0,0,0,1,0,0,cos(theta1),0;
                                    0,0,0,0,0,1,-1,0,0];

        p1.e = transpose(Jcm_link1) * p2.e;
        b =  Jcm_link1 * p1.f;

        p2.f[1:3] = transpose(Rb_0)* b[1:3];
        p2.f[4:6] = transpose(Rb_0)* b[4:6];
```

G.7 MTF Jcm-link2

```
variables
        real Jcm_link2[6,9], theta1, theta2, theta3;
        real global Ycm1,Xcm1,k, h,L1,n;
        real global Rb_0[3,3];
        real a[6], b[6];
equations

        theta1 = jointAngles[1];
        theta2 = jointAngles[2];
        theta3 = jointAngles[3];
```

```
        Jcm_link2 = [1, 0, 0, 0, -h - sin(theta2)*L1 + (-sin(theta2)*cos(theta3) -
        ↪  cos(theta2)*sin(theta3))*n, sin(theta1)*cos(theta2)*L1 -
        ↪  (-sin(theta1)*cos(theta2)*cos(theta3) + sin(theta1)*sin(theta2)*sin(theta3))*n,
        ↪  -sin(theta1)*cos(theta2)*L1 + (-sin(theta1)*cos(theta2)*cos(theta3) +
        ↪  sin(theta1)*sin(theta2)*sin(theta3))*n, cos(theta1)*(-sin(theta2)*L1 +
        ↪  (-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*n),
        ↪  cos(theta1)*(-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*n;
                                    0, 1, 0, h + sin(theta2)*L1 - (-sin(theta2)*cos(theta3) -
                                    ↪  cos(theta2)*sin(theta3))*n, 0,
                                    ↪  cos(theta1)*cos(theta2)*L1 +
                                    ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                    ↪  cos(theta1)*sin(theta2)*sin(theta3))*n,
                                    ↪  -cos(theta1)*cos(theta2)*L1 -
                                    ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                    ↪  cos(theta1)*sin(theta2)*sin(theta3))*n,
                                    ↪  -sin(theta1)*(-sin(theta2)*L1 +
                                    ↪  (-sin(theta2)*cos(theta3) -
                                    ↪  cos(theta2)*sin(theta3))*n),
                                    ↪  -sin(theta1)*(-sin(theta2)*cos(theta3) -
                                    ↪  cos(theta2)*sin(theta3))*n;
                                    0, 0, 1, -sin(theta1)*cos(theta2)*L1 +
                                    ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                    ↪  sin(theta1)*sin(theta2)*sin(theta3))*n,
                                    ↪  -cos(theta1)*cos(theta2)*L1 -
                                    ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                    ↪  cos(theta1)*sin(theta2)*sin(theta3))*n, 0, 0,
                                    ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*L1 +
                                    ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                    ↪  sin(theta1)*sin(theta2)*sin(theta3))*n) -
                                    ↪  cos(theta1)*(cos(theta1)*cos(theta2)*L1 +
                                    ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                    ↪  cos(theta1)*sin(theta2)*sin(theta3))*n),
                                    ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*cos(theta3) +
                                    ↪  sin(theta1)*sin(theta2)*sin(theta3))*n -
                                    ↪  cos(theta1)*(cos(theta1)*cos(theta2)*cos(theta3) -
                                    ↪  cos(theta1)*sin(theta2)*sin(theta3))*n;
                                    0, 0, 0, 1, 0, 0, 0, sin(theta1), sin(theta1);
                                    0, 0, 0, 0, 1, 0, 0, cos(theta1), cos(theta1);
                                    0, 0, 0, 0, 0, 1, -1, 0, 0];
        a[1:3] = Rb_0 * p2.e[1:3];
        a[4:6] = Rb_0 * p2.e[4:6];
        p1.e = transpose(Jcm_link2) * p2.e;
        b = Jcm_link2 * p1.f;

        p2.f[1:3] = transpose(Rb_0)* b[1:3];
        p2.f[4:6] = transpose(Rb_0)* b[4:6];

```

G.8 MTF J-tip

```
 variables
        real J_tip[6,9], theta1, theta2, theta3;
        real global Ycm1,Xcm1,k, h,L1,L2,n;
        real global Rb_0[3,3];
        real a[6], b[6];
equations
```

```
7
8         theta1 = jointAngles[1];
9         theta2 = jointAngles[2];
10        theta3 = jointAngles[3];
11
12        J_tip = [1, 0, 0, 0, -h - sin(theta2)*L1 + (-sin(theta2)*cos(theta3) -
          ↪  cos(theta2)*sin(theta3))*L2, sin(theta1)*cos(theta2)*L1 -
          ↪  (-sin(theta1)*cos(theta2)*cos(theta3) + sin(theta1)*sin(theta2)*sin(theta3))*L2,
          ↪  -sin(theta1)*cos(theta2)*L1 + (-sin(theta1)*cos(theta2)*cos(theta3) +
          ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2, cos(theta1)*(-sin(theta2)*L1 +
          ↪  (-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*L2),
          ↪  cos(theta1)*(-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*L2;
13                              0, 1, 0, h + sin(theta2)*L1 - (-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2, 0, cos(theta1)*cos(theta2)*L1 +
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -cos(theta1)*cos(theta2)*L1 -
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -sin(theta1)*(-sin(theta2)*L1 + (-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2),
                                ↪  -sin(theta1)*(-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2;
14                              0, 0, 1, -sin(theta1)*cos(theta2)*L1 +
                                ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -cos(theta1)*cos(theta2)*L1 -
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2, 0, 0,
                                ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*L1 +
                                ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2) -
                                ↪  cos(theta1)*(cos(theta1)*cos(theta2)*L1 +
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2),
                                ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2 -
                                ↪  cos(theta1)*(cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2;
15                              0, 0, 0, 1, 0, 0, 0, sin(theta1), sin(theta1);
16                             0, 0, 0, 0, 1, 0, 0, cos(theta1), cos(theta1);
17                             0, 0, 0, 0, 0, 1, -1, 0, 0];
18
19        a[1:3] = Rb_0 * p2.e[1:3];
20        a[4:6] = Rb_0 * p2.e[4:6];
21        p1.e = transpose(J_tip) * a;
22        b = J_tip * p1.f;
23
24        p2.f[1:3] = transpose(Rb_0)* b[1:3];
25        p2.f[4:6] = transpose(Rb_0)* b[4:6];
26
27
28
29
```

H Wire modoluses

```
variables
        real denominator;
        real x1,y1,z1,x2,y2,z2;
        real mx1,my1,mz1,mx2,my2,mz2;
equations

        x1 = TipPosition[1];
        y1 = TipPosition[2];
        z1 = TipPosition[3];

        x2 = SpreaderPosition[1];
        y2 = SpreaderPosition[2];
        z2 = SpreaderPosition[3];

mx1 = 0.5*(-2*x2 + 2*x1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
mx2 = 0.5*(2*x2 - 2*x1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
my1 = 0.5*(-2*y2 + 2*y1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
my2 = 0.5*(2*y2 - 2*y1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
mz1 = 0.5*(-2*z2 + 2*z1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);
mz2 = 0.5*(2*z2 - 2*z1)/sqrt((x2 - x1)^2 + (y2 - y1)^2 + (z2 - z1)^2);


        TipModoluses = [mx1;
                                              my1;
                                              mz1];


        SpreaderModoluses = [mx2;
                                                        my2;
                                                        mz2];
```

I MR-wire

```

parameters
        real tsi = 5;                                   // Realtive damping factor
        real MassOfWire = 10; // [kg]
variables
        real c;
equations
        c = 2*tsi*sqrt(inputSignal_wireStiffness*MassOfWire); // Damping coefficent
        p.e = c * p.f;
```

J C-wire

```
parameters
        real global initalWireLength;
        real global initial[3,1];
        real global E_wire; //Wire E-module [Pa]
        real global D_wire;//Diameter of wire [m]

variables
```

```
        real initialLength[1];
        real k_wire;
        real totalWireLength;
initialequations
        totalWireLength = initalWireLength;
equations

        totalWireLength = int(p.f,initalWireLength);

        k_wire = E_wire * pi*(D_wire/2)^2/totalWireLength;
  state = int(p.f);
  p.e = k_wire*state;

        outputSignal_wireStiffness = k_wire;


```

K Integrator- spreaderPosition

```
parameters
        real global initial[3,1];               // initial value
        real global initalWireLength;
        real global H;
        real global Draught;
        real global vcg_ship;
variables
        real initialSpreaderPosition[3];
        real craneTip_init[3];
        real craneBase[3];
        real craneLink1[3];
        real craneLink2[3];
        real global r0_b_cog[3];
        real global r0_cog_0[3];
        real global Rb_0[3,3], Rb_1[3,3], R1_2[3,3], R2_3[3,3];

        real global rb_1_b[3];
        real global r1_2_1[3];
        real global r2_3_2[3];
        real global r3_8_3[3];
        real global theta1_rad_init,theta2_rad_init,theta3_rad_init;
        real init_phi_rad;
         real init_theta_rad;
         real init_psi_rad;
        real global initialVesselAngles[3];
initialequations

        init_phi_rad = initialVesselAngles[1];
        init_theta_rad = initialVesselAngles[2];
        init_psi_rad = initialVesselAngles[3];

        Rb_0 = [cos(init_psi_rad)*cos(init_theta_rad), sin(init_psi_rad)*cos(init_theta_rad),
        ↪  -sin(init_theta_rad);
```

```
                                -sin(init_psi_rad)*cos(init_phi_rad) +
                                ↪  cos(init_psi_rad)*sin(init_theta_rad)*sin(init_phi_rad),
                                ↪  cos(init_psi_rad)*cos(init_phi_rad) +
                                ↪  sin(init_psi_rad)*sin(init_theta_rad)*sin(init_phi_rad),
                                ↪  cos(init_theta_rad)*sin(init_phi_rad);
                                sin(init_psi_rad)*sin(init_phi_rad) +
                                ↪  cos(init_psi_rad)*sin(init_theta_rad)*cos(init_phi_rad),
                                ↪  -cos(init_psi_rad)*sin(init_phi_rad) +
                                ↪  sin(init_psi_rad)*sin(init_theta_rad)*cos(init_phi_rad),
                                ↪  cos(init_theta_rad)*cos(init_phi_rad)];

        Rb_1 = [sin(theta1_rad_init), cos(theta1_rad_init), 0;
                        cos(theta1_rad_init ), -sin(theta1_rad_init ), 0;
                        0, 0, -1];
        R1_2 = [1, 0, 0;
                         0, cos(theta2_rad_init), -sin(theta2_rad_init);
                          0, sin(theta2_rad_init), cos(theta2_rad_init)];
        R2_3 = [1, 0, 0;
               0, cos(theta3_rad_init), -sin(theta3_rad_init);
               0, sin(theta3_rad_init), cos(theta3_rad_init)];

        r0_b_cog = [sin(0)*(H - vcg_ship); sin(0)*(H - vcg_ship);(H - vcg_ship)]
        ↪  +[0;0;vcg_ship-Draught];
        craneTip_init = r0_b_cog + transpose(Rb_0)*rb_1_b +
        ↪  transpose(Rb_0)*transpose(Rb_1)*r1_2_1 + transpose(Rb_0)*transpose(Rb_1)*R1_2*r2_3_2
        ↪  + transpose(Rb_0)*transpose(Rb_1)*R1_2*R2_3*r3_8_3;
        initialSpreaderPosition = craneTip_init - [0;0;initalWireLength];
        SpreaderPositionZ = initialSpreaderPosition[3];
equations

        output = int (input, initialSpreaderPosition);
   SpreaderPositionZ = output[3];

```

L IC - spreader

```
    parameters

        real global Mass;
        real global r_CG_Spreader[3,1];
        real I_bb [3,3] = [0,0,0;
                                                                0,0,0;
                                                                0,0,0];
variables
        // Mass matrix of spreader
        real global M[6,6];
        real global Minv[6,6];

        // Vectors to calculate generalized momentum of spreader
        real p_tot [6,1];
        real pIf_tot [6,1];
        real pCe_tot [6,1];
        real p_tot1[3], p_tot2[3];

 // Matrix containing rigid bbody centripetal and coriolies matrix
        real CRB [6,6];
```

```

initialequations

        I_bb = M[4:6,4:6] - Mass*skew(r_CG_Spreader)*skew(r_CG_Spreader);
equations

        p_tot1 = int(pIv.e);
        p_tot2 = int(pIw.e);

        p_tot[1:3] = p_tot1;
        p_tot[4:6] = p_tot2;
        pIf_tot = Minv*p_tot;

        CRB[1:3,1:3] = [0,0,0;
                                                0,0,0;
                                                0,0,0];
        CRB[1:3,4:6] = -Mass*skew(pIv.f) - Mass*skew(skew(pIw.f)*r_CG_Spreader);
        CRB[4:6,1:3] = -Mass*skew(pIv.f) - Mass*skew(skew(pIw.f)*r_CG_Spreader);
        CRB[4:6,4:6] =  Mass*skew(skew(pIv.f)*r_CG_Spreader) - skew(I_bb*pIw.f);

        pCe_tot = -CRB*pIf_tot;

  // Setting output
        pIv.f = pIf_tot[1:3];
        pCv.e = pCe_tot[1:3];
        pIw.f = pIf_tot[4:6];
        pCw.e = pCe_tot[4:6];


```

L.1 R-element rotational wind resistance

```
parameters
        real r[3,3] = 0*[100.0, 0.0, 0.0; 0.0, 100.0, 0.0; 0.0, 0.0, 100.0];
equations
        p.e = r * p.f;

```

L.2 R-element translational wind resistance

```
parameters
        real r[3,3] = 0*[5.0, 0.0, 0.0; 0.0, 5.0, 0.0; 0.0, 0.0, 5.0];
equations
        p.e = r * p.f;

```

M Code for control system

M.1 FSM

```


```

```
variables
        real CargoTransport;
        real Standby;
initialequations

        // Setting intial
        Standby = FALSE ;
        CargoTransport = TRUE;
        InitiateCraneTransport = FALSE;
        InitiateSwayComp = FALSE;
        InitiateStationKeeping = FALSE;
        InitiateWireControl = FALSE;

equations



        if Standby then
                CargoTransport = FALSE;
        end;


        if CargoTransport == TRUE then

                if EndEffectorPointReached == FALSE then
                        InitiateCraneTransport = TRUE;
                 elsif EndEffectorPointReached == TRUE then
                        InitiateSwayComp = TRUE;
                        InitiateStationKeeping = TRUE;
                        InitiateCraneTransport = FALSE;

                        if SwayErrorMarginReached == TRUE then
                                        InitiateSwayComp = FALSE;
                                        InitiateWireControl = TRUE;

                                if CargoTuchdown == TRUE then
                                        CargoTransport = FALSE;
                                        Standby = TRUE;
                                end;

                        end;

                 end;

        end;
```

N MTF translational motion to crane joints

```
 variables
        real J_tip[6,9], theta1, theta2, theta3;
        real global Ycm1,Xcm1,k, h,L1,L2,n;
```

```
        real global R0_b[3,3];
        real a[3], b[3];
        real global Rb_0[3,3];
equations

        theta1 = jointAngles[1];
        theta2 = jointAngles[2];
        theta3 = jointAngles[3];

        J_tip = [1, 0, 0, 0, -h - sin(theta2)*L1 + (-sin(theta2)*cos(theta3) -
        ↪  cos(theta2)*sin(theta3))*L2, sin(theta1)*cos(theta2)*L1 -
        ↪  (-sin(theta1)*cos(theta2)*cos(theta3) + sin(theta1)*sin(theta2)*sin(theta3))*L2,
        ↪  -sin(theta1)*cos(theta2)*L1 + (-sin(theta1)*cos(theta2)*cos(theta3) +
        ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2, cos(theta1)*(-sin(theta2)*L1 +
        ↪  (-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*L2),
        ↪  cos(theta1)*(-sin(theta2)*cos(theta3) - cos(theta2)*sin(theta3))*L2;
                                0, 1, 0, h + sin(theta2)*L1 - (-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2, 0, cos(theta1)*cos(theta2)*L1 +
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -cos(theta1)*cos(theta2)*L1 -
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -sin(theta1)*(-sin(theta2)*L1 + (-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2),
                                ↪  -sin(theta1)*(-sin(theta2)*cos(theta3) -
                                ↪  cos(theta2)*sin(theta3))*L2;
                                0, 0, 1, -sin(theta1)*cos(theta2)*L1 +
                                ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2,
                                ↪  -cos(theta1)*cos(theta2)*L1 -
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2, 0, 0,
                                ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*L1 +
                                ↪  (-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2) -
                                ↪  cos(theta1)*(cos(theta1)*cos(theta2)*L1 +
                                ↪  (cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2),
                                ↪  sin(theta1)*(-sin(theta1)*cos(theta2)*cos(theta3) +
                                ↪  sin(theta1)*sin(theta2)*sin(theta3))*L2 -
                                ↪  cos(theta1)*(cos(theta1)*cos(theta2)*cos(theta3) -
                                ↪  cos(theta1)*sin(theta2)*sin(theta3))*L2;
                                0, 0, 0, 1, 0, 0, 0, sin(theta1), sin(theta1);
                               0, 0, 0, 0, 1, 0, 0, cos(theta1), cos(theta1);
                               0, 0, 0, 0, 0, 1, -1, 0, 0];

        a = Rb_0*p1.f;
        b = transpose(Rb_0)*p2.e;

        p1.e = J_tip[1:3,7:9]*b;
        p2.f = inverse(J_tip[1:3,7:9])*a;
```

N.1 Station-keeping controllers for crane-tip x,y,z

```
    parameters
        real K = 1 {};                          // Proportional gain
        real Td = 1.0 {s};                  // Derivative time constant: Td > 0
        real N = 1 {};                          // Derivative gain limitation.
        real Ti = 1.0 {s};                          // Integral time constant: Ti > 0
        real minI = -2;
        real maxI = 2;
variables
        real error;
        real hidden uP,uI,uD,uDstate;
equations
        error = SP - MV;
        uP = K * error;
        uI = limint ((K * error / Ti), minI, maxI);
        uDstate = int (uD * N / Td);
        uD = K * error * N - uDstate;
        output = uP + uD + uI;
```

N.2 Sway controllers x,y

```
    parameters
        real K = 1 {};                          // Proportional gain
        real Td = 1.0 {s};                  // Derivative time constant: Td > 0
        real N = 1 {};                          // Derivative gain limitation.
        real Ti = 1.0 {s};                          // Integral time constant: Ti > 0
        real minI = -2;
        real maxI = 2;
variables
        real error;
        real hidden uP,uI,uD,uDstate;
equations
        error = SP - MV;
        uP = K * error;
        uI = limint ((K * error / Ti), minI, maxI);
        uDstate = int (uD * N / Td);
        uD = K * error * N - uDstate;
        output = uP + uD + uI;
```

N.3 Reference filter

```

parameters
        real zeta = 1;
        real omega = 2*pi/10;
        real global wireLength;
variables
        real OMEGA;
        real GAMMA;
        real A_f;
        real t;
        real x_ref_dot;
        real x_d;
```

```
        real v_d;
        real a_d;
        real x_ref;
        real initalWirePosition;
initialequations
        t = 1/omega;
        A_f = 1/t;
        OMEGA = 2*zeta*omega;
        GAMMA = omega^2;
        x_ref_dot = 0;
        x_d = 0; v_d = 0; a_d = 0;
        initalWirePosition = -wireLength;
equations


        // first order
        //x_ref = eta_d/A_f - int(x_ref_dot)/A_f;
        x_ref_dot = -int(x_ref_dot)*A_f + eta_d*A_f;
        x_ref = int(x_ref_dot);
        //third order

        a_d = x_ref*GAMMA - OMEGA*v_d - GAMMA*x_d;
        v_d = int(a_d);
        x_d = int(v_d);
         referenceSignal = x_d;
```

N.4 Wire lowering and heave compensation PID controller

```
    parameters
        real K = 3 {};                      // Proportional gain
        real Td = 1.0 {s};                // Derivative time constant: Td > 0
        real N = 1 {};                      // Derivative gain limitation.
        real Ti = 1.0 {s};                      // Integral time constant: Ti > 0
        real minI = -2;
        real maxI = 2;
variables
        real error;
        real hidden uP,uI,uD,uDstate;
equations
        error = SP - MV;
        uP = K * error;
        uI = limint ((K * error / Ti), minI, maxI);
        uDstate = int (uD * N / Td);
        uD = K * error * N - uDstate;
        output = uP + uD + uI;
```

O Inverse kinematics

```

/* Equation Submodel
Enter your equations here. You can use the Toolbar buttons at the top ( Add , f(x) etc. ).
*/
parameters
 real global H;
```

```
variables
        real thetha1_desired, thetha2_desired, thetha3_desired;
        real global L1,L2, r1_2_1[3];
        real c3,c2,s3,s2;

equations









        //Computing desdired joint angles
        c3 = (desiredPosition[1]^2 + desiredPosition[2]^2 + desiredPosition[3]^2 -
        ↪  L1^2-L2^2)/(2*L1*L2);
        s3 = -sqrt(1-c3^2);
        s2 = ((L1 + L2*c3)*desiredPosition[3]-L2*s3*sqrt(desiredPosition[1]^2 +
        ↪  desiredPosition[2]^2))/(desiredPosition[1]^2+ desiredPosition[2]^2
        ↪  +desiredPosition[3]^2);
        c2 = ((L1 + L2*c3)*sqrt(desiredPosition[1]^2 + desiredPosition[2]^2) +
        ↪  L2*s3*desiredPosition[3])/(desiredPosition[1]^2+ desiredPosition[2]^2
        ↪  +desiredPosition[3]^2);

        thetha1_desired = atan2(desiredPosition[2],desiredPosition[1]);

        thetha2_desired = atan2(s2,c2);

        thetha3_desired = atan2(s3,c3);

        desiredJointAngles[1] = thetha1_desired;
        desiredJointAngles[2] = thetha2_desired;
        desiredJointAngles[3] = thetha3_desired;
```

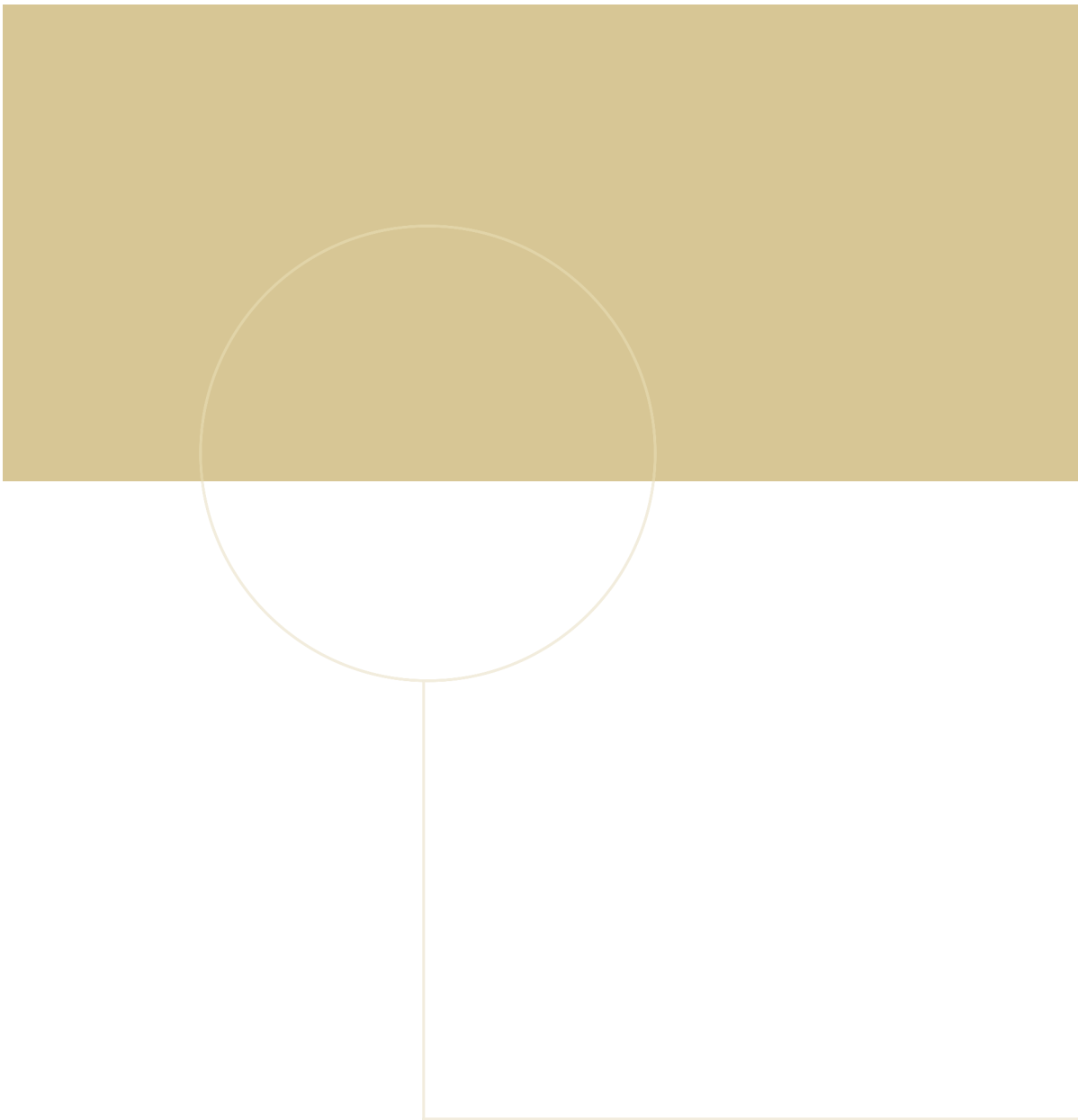
P Velocity profile generation

```
/* Equation Submodel
Enter your equations here. You can use the Toolbar buttons at the top ( Add , f(x) etc. ).
*/


//Same acceleration and velocity for all paths
parameters

        //Params for base
        //Initial velocity, accel
        real qi_dot_base = 0;
        real qi_ddot_base = 0.1;
        //End velocity, accel, time
        real qf_dot_base = 0;
        real qf_ddot_base = -0.05;
        real tf_base = 15;

        //Params for joint1
        //Initial velocity, accel
        real qi_dot_joint1 = 0;
        real qi_ddot_joint1 = 0.1;
        //End position,velocity, accel, time
```

```
        real qf_dot_joint1 = 0;
        real qf_ddot_joint1 = -0.1;
        real tf_joint1 = 10;


        //Params for joint2
        //Initial velocity, accel
        real qi_dot_joint2 = 0;
        real qi_ddot_joint2 = 0.1;
        //End velocity, accel, time
        real qf_dot_joint2 = 0;
        real qf_ddot_joint2 = -0.1;
        real tf_joint2 = 10;

variables
        real q1,q2,q3;                                                          // Joint
        ↪  positions
        real q1_dot,q2_dot,q3_dot;                  // Joint velocities
        real q1_ddot,q2_ddot,q3_ddot; // joint accelerations
        real a10,a11,a12,a13,a14,a15;
        real a20,a21,a22,a23,a24,a25;
        real a30,a31,a32,a33,a34,a35;
        real global theta1_rad_init, theta2_rad_init, theta3_rad_init;

        // inital and end positions of all three joints
        real qi_base, qf_base;
        real qi_joint1, qf_joint1;
        real qi_joint2, qf_joint2;

        // time param
        real activationTime;
        real t_relative;
initialequations

                activationTime = -1;
                t_relative = 0;
equations

        //Setting initial and and rotations

        qi_base = theta1_rad_init;
        qf_base = desiredJointAngles[1];
        qi_joint1 = theta2_rad_init;
        qf_joint1 = desiredJointAngles[2];
        qi_joint2 = theta3_rad_init;
        qf_joint2 = desiredJointAngles[3];

        if InitializeTransport == TRUE then

                if activationTime < 0 then
                        activationTime = time;
                else
                        t_relative =         time - activationTime;
                end;


                //Params for base
                a10 = qi_base;
```

```
80          a11 = qi_dot_base;
81          a12 = qi_ddot_base/2;
82          a13 = (qf_ddot_base*tf_base^2 - 3*qi_ddot_base*tf_base^2 - 8*qf_dot_base*tf_base
            ↪  - 12*qi_dot_base*tf_base + 20*qf_base - 20*qi_base)/(2*tf_base^3);
83          a14 = -(2*qf_ddot_base*tf_base^2 - 3*qi_ddot_base*tf_base^2 -
            ↪  14*qf_dot_base*tf_base - 16*qi_dot_base*tf_base + 30*qf_base -
            ↪  30*qi_base)/(2*tf_base^4);
84          a15 = (qf_ddot_base*tf_base^2 - qi_ddot_base*tf_base^2 - 6*qf_dot_base*tf_base -
            ↪  6*qi_dot_base*tf_base + 12*qf_base - 12*qi_base)/(2*tf_base^5);
85
86          // Params for lower joint
87          a20 = qi_joint1;
88          a21 = qi_dot_joint1;
89          a22 = qi_ddot_joint1/2;
90          a23 = (qf_ddot_joint1*tf_joint1^2 - 3*qi_ddot_joint1*tf_joint1^2 -
            ↪  8*qf_dot_joint1*tf_joint1 - 12*qi_dot_joint1*tf_joint1 + 20*qf_joint1 -
            ↪  20*qi_joint1)/(2*tf_joint1^3);
91          a24 = -(2*qf_ddot_joint1*tf_joint1^2 - 3*qi_ddot_joint1*tf_joint1^2 -
            ↪  14*qf_dot_joint1*tf_joint1 - 16*qi_dot_joint1*tf_joint1 + 30*qf_joint1 -
            ↪  30*qi_joint1)/(2*tf_joint1^4);
92          a25 = (qf_ddot_joint1*tf_joint1^2 - qi_ddot_joint1*tf_joint1^2 -
            ↪  6*qf_dot_joint1*tf_joint1 - 6*qi_dot_joint1*tf_joint1 + 12*qf_joint1 -
            ↪  12*qi_joint1)/(2*tf_joint1^5);
93
94          // Params for upper joint
95          a30 = qi_joint2;
96          a31 = qi_dot_joint2;
97          a32 = qi_ddot_joint2/2;
98          a33 = (qf_ddot_joint2*tf_joint2^2 - 3*qi_ddot_joint2*tf_joint2^2 -
            ↪  8*qf_dot_joint2*tf_joint2 - 12*qi_dot_joint2*tf_joint2 + 20*qf_joint2 -
            ↪  20*qi_joint2)/(2*tf_joint2^3);
99          a34 = -(2*qf_ddot_joint2*tf_joint2^2 - 3*qi_ddot_joint2*tf_joint2^2 -
            ↪  14*qf_dot_joint2*tf_joint2 - 16*qi_dot_joint2*tf_joint2 + 30*qf_joint2 -
            ↪  30*qi_joint2)/(2*tf_joint2^4);
100         a35 = (qf_ddot_joint2*tf_joint2^2 - qi_ddot_joint2*tf_joint2^2 -
            ↪  6*qf_dot_joint2*tf_joint2 - 6*qi_dot_joint2*tf_joint2 + 12*qf_joint2 -
            ↪  12*qi_joint2)/(2*tf_joint2^5);
101
102
103         if t_relative <= tf_base then
104                 q1 = a15*t_relative^5 + a14*t_relative^4 + a13*t_relative^3 +
                    ↪  a12*t_relative^2 + a11*t_relative + a10;
105                 q1_dot = 5*a15*t_relative^4 + 4*a14*t_relative^3 + 3*a13*t_relative^2 +
                    ↪  2*a12*t_relative + a11;
106                 q1_ddot = 20*a15*t_relative^3 + 12*a14*t_relative^2 + 6*a13*t_relative +
                    ↪  2*a12;
107                 velocityReferences[1] = q1_dot;
108         else
109                 velocityReferences[1] = 0;
110                 q1 = 0;
111                 q1_dot = 0;
112                 q1_ddot = 0;
113
114         end;
115
116         if t_relative <= tf_joint1 then
117                 q2 = a25*t_relative^5 + a24*t_relative^4 + a23*t_relative^3 +
                    ↪  a22*t_relative^2 + a21*t_relative + a20;
```

```
118                     q2_dot = 5*a25*t_relative^4 + 4*a24*t_relative^3 + 3*a23*t_relative^2 +
                        ↪  2*a22*t_relative + a21;
119                     q2_ddot = 20*a25*t_relative^3 + 12*a24*t_relative^2 + 6*a23*t_relative +
                        ↪  2*a22;
120                     velocityReferences[2] = q2_dot;
121             else
122                     velocityReferences[2] = 0;
123                     q2 = 0;
124                     q2_dot = 0;
125                     q2_ddot = 0;
126             end;
127
128             if t_relative <= tf_joint2 then
129                     q3 = a35*t_relative^5 + a34*t_relative^4 + a33*t_relative^3 +
                        ↪  a32*t_relative^2 + a31*t_relative + a30;
130                     q3_dot = 5*a35*t_relative^4 + 4*a34*t_relative^3 + 3*a33*t_relative^2 +
                        ↪  2*a32*t_relative + a31;
131                     q3_ddot = 20*a35*t_relative^3 + 12*a34*t_relative^2 + 6*a33*t_relative +
                        ↪  2*a32;
132                     velocityReferences[3] = q3_dot;
133             else
134                     velocityReferences[3] = 0;
135                     q3 = 0;
136                     q3_dot = 0;
137                     q3_ddot = 0;
138             end;
139
140     else
141             q1 = 0;
142             q2 = 0;
143             q3 = 0;
144             q1_dot = 0;
145             q2_dot = 0;
146             q3_dot = 0;
147             q1_ddot = 0;
148             q2_ddot = 0;
149             q3_ddot = 0;
150     end;
```



NTNU

Norwegian University of
Science and Technology